University of Alberta

SEQUENCE-BASED PROTEIN FUNCTION PREDICTION

by

Brett Poulin

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

in

Department of Computing Science

Edmonton, Alberta
Fall 2004

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 0-612-95833-7
Our file Notre référence
ISBN: 0-612-95833-7

The author has granted a non-exclusive license allowing the Library and Archives Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou aturement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

We have not succeeded in answering all your problems. The answers that we have found only serve to raise a whole new set of questions. In some ways we feel we are as confused as ever, but we believe we are confused on a higher level, and about more important things. (From the final report of an anonymous industrial consultant, quoted by A.T. Winfree in "When Time Breaks Down".)

For Ben, Abby, and especially Jen, who make everything more fun.

Acknowledgements

I wish to thank my wife, Jen, and our children, Ben and Abby, for inspiration every day. I appreciate the continuing encouragement and support of my parents. I would also like to thank my supervisors, Duane Szafron and Russ Greiner, for giving me valuable advice, encouragement, and independence. I am also grateful to everyone in the bioinformatics research group for terrific discussions (research-related and otherwise).

Contents

1 Introduction **1**
- 1.1 The Central Dogma of Molecular Biology . . . 1
- 1.2 Protein Sequence, Structure, and Function . . . 3
- 1.3 Protein Classes . . . 6
- 1.4 Related Work In Protein Function Prediction . . . 10
  - 1.4.1 Protein Function Determination . . . 10
  - 1.4.2 Protein Function Prediction . . . 10
    - Sequence Alignments . . . 11
    - Hidden Markov Models . . . 11
    - Pattern Databases . . . 11
    - Proteome Analyst . . . 12
  - 1.4.3 Limitations . . . 12
- 1.5 Research Goal . . . 13
- 1.6 Summary . . . 14

2 Machine Learning **15**
- 2.1 Introduction . . . 15
- 2.2 The Classification Problem . . . 16
- 2.3 Model Selection . . . 16
  - 2.3.1 Performance and Error . . . 16
  - 2.3.2 Hold-Out and Cross-Validation . . . 18
  - 2.3.3 Overfitting . . . 19
  - 2.3.4 Performance Measures . . . 20
- 2.4 Examples of Classification Techniques . . . 27
  - 2.4.1 Nearest-Neighbor . . . 27
  - 2.4.2 Decision Trees . . . 28
  - 2.4.3 Naïve Bayes . . . 32
  - 2.4.4 Support Vector Machines . . . 34

3 Sequence Patterns **38**
- 3.1 Sequence Patterns and Protein Function . . . 38

3.1.1 Deterministic Patterns . . . 41
3.1.2 Probabilistic Patterns . . . 43
3.2 Deterministic Sequence Patterns . . . 44
3.2.1 Common Substrings . . . 44
3.2.2 Consensus Sequences . . . 45
3.2.3 Alignments . . . 46
Pairwise Alignments . . . 46
Multiple Alignments . . . 47
3.2.4 Regular Languages . . . 49
3.3 Probabilistic Sequence Patterns . . . 53
3.3.1 Probabilistic Profiles . . . 53
3.3.2 Markov Models . . . 57
Markov Chains . . . 57
Hidden Markov Models . . . 62
3.4 Kernel Methods . . . 72
3.5 Summary . . . 72

4 Probabilistic Suffix Trees **74**
4.1 Suffix Trees . . . 74
4.1.1 Building and Using Suffix Trees . . . 76
4.1.2 Linear Time Construction of Suffix Trees . . . 79
4.2 Probabilistic Suffix Trees . . . 81
4.2.1 Variable Length Markov Models (VMMs) . . . 81
4.2.2 Probabilistic Suffix Trees . . . 83
Prediction with a bPST . . . 83
Building a bPST . . . 86
4.2.3 PSTs vs HMMs . . . 86
4.3 Linear Time Construction and Prediction with PSTs . . . 87
4.3.1 Building an Efficient PST . . . 88
4.3.2 Prediction with an Efficient PST . . . 90
4.3.3 Pruning . . . 92
4.3.4 Smoothing . . . 94
4.3.5 Equivalence of PST Implementations . . . 96

5 Experiments for Efficient Protein Function Prediction **101**
5.1 Datasets . . . 101
5.2 Evaluation . . . 103
5.3 Alignment-Based Methods . . . 105
5.3.1 BLAST . . . 105
5.3.2 Proteome Analyst . . . 110
5.4 Pattern-based methods . . . 111

5.4.1 HMMer 111
5.4.2 Pfam and PROSITE Pattern Databases 113
5.4.3 Probabilistic Suffix Trees 115
bPSTs 115
ePSTs 116
5.5 Comparison of Time Requirements 124
5.6 Summary 132

6 Conclusion **134**
6.1 Discussion of Results 134
6.2 Future Work 136
6.2.1 Experimentation with PSTs 136
6.2.2 Improvements to PSTs 137
6.2.3 New Methods Related to PSTs 138
6.3 Summary 138

Bibliography **139**

A Supplementary Results **148**

B Supplementary Tables and Figures **173**

C Backoff Smoothing **179**

List of Tables

1.1 The Nucleic Acids 2
1.2 The Amino Acids and the Genetic Code 4

2.1 Confusion Matrix 23
2.2 Multi-class Confusion Matrix 25
2.3 Example Machine Learning Data 29
2.4 Example SVM Data 35

3.1 Markov Chain Probabilities 59

5.1 Majority Classification 104
5.2 BLAST Nearest-Neighbor 106
5.3 Proteome Analyst 111
5.4 HMMer 112
5.5 Pfam and PROSITE with Naïve Bayes 113
5.6 Pfam and PROSITE with SVM 114
5.7 bPST 116
5.8 ePST 117
5.9 ePST Local Predictions 119
5.10 ePSTs Evaluated at Maximum F-measure 120
5.11 ePST Local without Smoothing 121
5.12 ePST Local with Shorter History Length 122
5.13 ePST Local with Shorter History Length 123
5.14 Comparison of Training Time Requirements Dependent on Training Set Size 125
5.15 Comparison of Prediction Time Requirements Dependent on Training Set Size 126
5.16 Comparison of Prediction Time Requirements Dependent on Test Set Size 127
5.17 Non-Monotonic Increase in bPST Training Time Requirements Dependent on Training Set Size 129
5.18 Dependence of ePST Testing Time on Maximum History Length . . 130

5.19 Dependence of ePST Testing Time on Maximum History Length without the α Calculation 132

A.1 Results for K^+ Channels, Majority Classifier 149
A.2 Results for Gram- Subcell, Majority Classifier 149
A.3 Results for GO with SwissProt 1/4, Majority Classifier 150
A.4 Results for K^+ Channels, BLAST Nearest-Neighbor 151
A.5 Results for Gram- Subcell, BLAST Nearest Neighbor 151
A.6 Results for GO with SwissProt 1/4, BLAST Nearest-Neighbor . . . 152
A.7 Results for K^+ Channels, Proteome Analyst 153
A.8 Results for Gram- Subcell, Proteome Analyst 153
A.9 Results for GO with SwissProt 1/4, Proteome Analyst 154
A.10 Results for K^+ Channels, HMMer 155
A.11 Results for Gram- Subcell, HMMer 156
A.12 Results for K^+ Channels, Pfam and PROSITE with Naïve Bayes . . 156
A.13 Results for Gram- Subcell, Pfam and PROSITE with Naïve Bayes . 157
A.14 Results for GO with SwissProt 1/4, Pfam and PROSITE with Naïve Bayes 158
A.15 Results for K^+ Channels, Pfam and PROSITE with SVM 159
A.16 Results for Gram- Subcell, Pfam and PROSITE with SVM 159
A.17 Results for GO with SwissProt 1/4, Pfam and PROSITE with SVM . 160
A.18 Results for K^+ Channels, bPST 161
A.19 Results for Gram- Subcell, bPST 162
A.20 Results for GO with SwissProt 1/4, bPST 163
A.21 Results for K^+ Channels, Global ePST at Isopoint 164
A.22 Results for Gram- Subcell, Global ePST at Isopoint 164
A.23 Results for GO with SwissProt 1/4, Global ePST at Isopoint 165
A.24 Results for K^+ Channels, ePST at Isopoint 166
A.25 Results for Gram- Subcell, ePST at Isopoint 166
A.26 Results for GO with SwissProt 1/4, ePST at Isopoint 167
A.27 Results for K^+ Channels, ePST at FMAX 40 168
A.28 Results for Gram- Subcell, Pfam and PROSITE with ePST FMAX 40 168
A.29 Results for GO with SwissProt 1/4, ePST at FMAX 40 169
A.30 Results for GO with SwissProt 1/4, ePST without Smoothing 170
A.31 Comparison over *pkinase* Pfam Family 171
A.32 Comparison over *metallothio* Pfam Family 171
A.33 Comparison of Mixed *pkinase* and *metallothio* Pfam Families 172

B.1 The 20 Amino Acids and Mappings to Smaller Alphabets. 176

List of Figures

1.1 The Central Dogma 2
1.2 Growth of Protein Sequence Databases 5
1.3 The Gene Ontology 7
1.4 A Gene Ontology Sample 8
1.5 Voltage-Gated Potassium (K^+) Channel Structure 9
1.6 The Research Goal 13

2.1 The Training and Classification Processes 17
2.2 Overfitting 19
2.3 Confusion in a Diagnostic Test 22
2.4 ROC Curves 25
2.5 Nearest-Neighbor Classification 27
2.6 A Decision Tree 28
2.7 A Separating Hyperplane 34

3.1 Example Protein Sequences 39
3.2 Zinc Finger Structure 41
3.3 Zinc Finger Sequence 42
3.4 Patterns in Example Sequences 42
3.5 Common Substrings 44
3.6 Consensus Sequences 45
3.7 Pairwise Alignment 47
3.8 Multiple Alignment 48
3.9 PROSITE Output 50
3.10 TEIRESIAS Output 51
3.11 PRATT Output 52
3.12 Position-Specific Scoring Matrix (PSSM) 54
3.13 Using a Position-Specific Scoring Matrix 55
3.14 Example Markov Chain Predictions 61
3.15 An Example HMM 64
3.16 HMMer Architecture. 68
3.17 GENSCAN HMM Architecture. 70

3.18 Pfam Domains of the HIV Protein GP120 71
3.19 Structure of the HIV Protein GP120 71
3.20 Example Pfam Results 73

4.1 A Suffix Tree 75
4.2 Building a Suffix Tree 77
4.3 Counting Substring Occurrences with a Suffix Tree 78
4.4 A Generalized Suffix Tree 78
4.5 Suffix Links 80
4.6 Bejerano and Yona's Probabilistic Suffix Tree 84
4.7 An Efficient Probabilistic Suffix Tree 89
4.8 Prediction with an ePST 91
4.9 Another Efficient Probabilistic Suffix Tree 97
4.10 Conversion of an ePST to a bPST 99
4.11 Efficient Probabilistic Suffix Tree 100

5.1 BLAST Coverage for 5-fold Cross-Validation, *SwissProt 1/4* 106
5.2 BLAST Nearest-Neighbor Performance Response to Varying E-value Threshold, *Gram- Subcell* 107
5.3 BLAST Nearest-Neighbor Performance Response to Varying E-value Threshold, *GO with SwissProt 1/4* 108
5.4 BLAST Coverage of SwissProt for Various Organisms 109
5.5 Comparison of Training Time Requirements Dependent on Training Set Size 126
5.6 Comparison of Testing Time Requirements Dependent on Testing Set Size 127
5.7 Comparison of Testing Time Requirements Dependent on Testing Set Size and Tree Size 131

B.1 Example Training Sequences - PSSM 174
B.2 Example Training Sequences - Markov Chain 175
B.3 Mapped Example Training Sequences 177
B.4 Conversion of an ePST to a bPST 178

Chapter 1

Introduction

1.1 The Central Dogma of Molecular Biology

In 1958, Francis Crick presented two revolutionary ideas to the British Society of Experimental Biology [29]. The 'Sequence Hypothesis' is the idea that the simple 4-letter alphabet of a DNA string (see Table 1.1) encodes the information necessary for the huge variety of proteins that form the structure and active components of life. The 'Central Dogma of Molecular Biology' is the idea that genetic information flows from DNA to RNA to protein. These concepts changed the way we look at life. Crick's proposals introduced a new paradigm to molecular biology – a paradigm with biological sequences at the core. The impact of this shift in thinking has extended beyond the field of molecular biology to reshape the way we see all of biology, medicine, disease, heredity and many other aspects of our everyday lives.

DNA, the genetic blueprint, contains the information in its simple sequence to encode the stunning variety of structural and catalytic molecules that are the stuff of life. DNA is transcribed into RNA, another sequence polymer that, among other things, may be processed and translated into an amino acid sequence (see Figure 1.1 and Table 1.2). The amino acid sequence is then folded[1] and processed to form an incredible variety of proteins, the main functional components of living cells.

Crick [29] underscored the importance of understanding how the simplicity of sequence information could be transformed into astounding structural and functional diversity.

> 'In the protein molecule, Nature has devised an instrument in which

[1] As a protein is created, the amino acids are extended in a string or sequence. This is the protein's primary structure. The amino acid sequence then becomes folded into a series of helices, sheets and coils called the secondary structure. These helices, sheets and coils are then further folded into the higher level or tertiary structure. Figure 1.5 shows how the helices and coils (secondary structure) of a potassium channel protein are organized relative to each other in order to form the tertiary structure. Quaternary structure defines the overall structure of groups of proteins that work together as a unit.

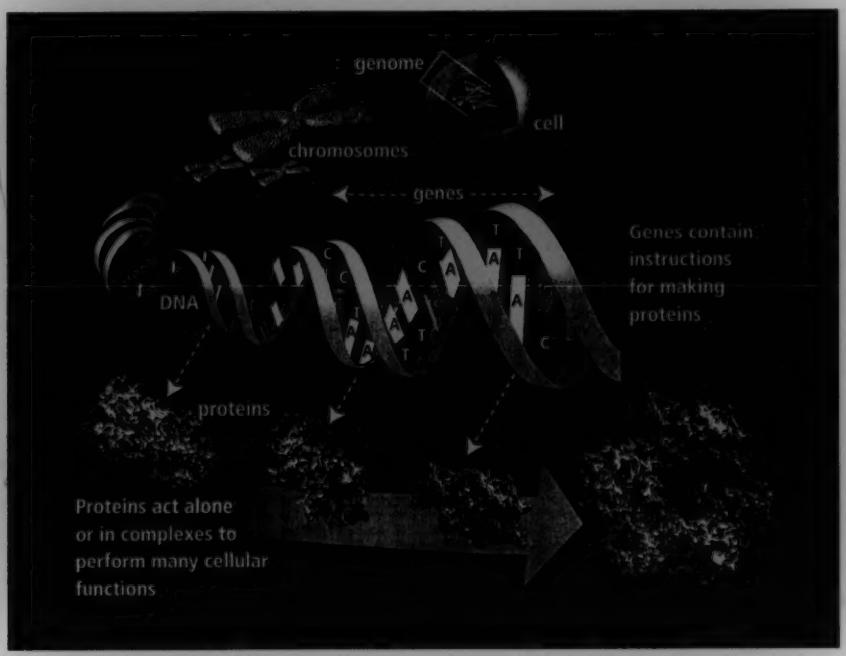
Table 1.1: The Nucleic Acids

The 4-letter DNA and 4-letter RNA alphabets. The nucleic acids are linear polymers. They are made up of sequences of bases that are connected by a backbone of deoxyribose sugars (ribose in the case of RNA) and phosphates. They naturally pair with the other bases, A with T/U and C with G.

DNA Base	Symbol	RNA Base	Symbol
Adenine	A	Adenine	A
Cytosine	C	Cytosine	C
Guanine	G	Guanine	G
Thymine	T	Uracil	U

Figure 1.1: The Central Dogma

An illustration of the flow of information from DNA to RNA to protein, coined by Francis Crick as the 'Central Dogma of Molecular Biology'. The image is courtesy of the U.S. Department of Energy Human Genome Program.



an underlying simplicity is used to express great subtlety and versatility; it is impossible to see molecular biology in proper perspective until this peculiar combination of virtues has been clearly grasped.'

Proteins are the machinery of life. Proteins make up structural and active elements of living cells. To eliminate proteins and their functions would be to end life as we know it. It is astonishing that so much of the complexity we see and experience in life can be encoded by relatively simple sequences of DNA, RNA and protein. The purpose of this work is to predict protein function from each protein's amino acid sequence.

1.2 Protein Sequence, Structure, and Function

Protein function has traditionally been determined through experimental means. The human and laboratory resources required to characterize the function of a single protein are substantial. When considering the vast number of proteins that are of potential research interest, these resource demands become prohibitive. Fortunately, there are other methods through which we can establish protein function for many new proteins.

The regions of biological sequences that are most important for life tend to be most conserved. Evolutionary pressure maintains the most essential patterns in DNA, RNA, and protein sequences while less crucial sequence regions tend to allow mutations to accumulate. In some cases, sequence regions that are not well conserved may be important for function but may also be flexible enough to allow some changes (the changes may even be beneficial). In most cases, these regions are less important for proper function and can be changed over generations without significant decrease in the fitness of the organisms in which they exist. For these reasons, conserved patterns in biological sequences can be important predictors of function.

Protein sequences in particular offer a wealth of information about cellular function. Proteins are the cell's main functional units. As a protein's function is determined by its structure and its structure is largely determined by its sequence, protein sequence patterns are very closely related to protein function. Because of this, sequence analysis can allow us to ascertain much of protein function by comparison of new sequences to those that have already been characterized.

In recent history, the amount of biological sequence information known has increased dramatically. The progress of sequencing technology and the many genome projects have generated vast amounts of DNA and RNA sequences. With that has come a flood of related protein information and in the 'post-genomic era' more and more attention now turns to proteins and their sequences. The recent growth of protein sequence databases shows the difficulty in keeping up with the increase

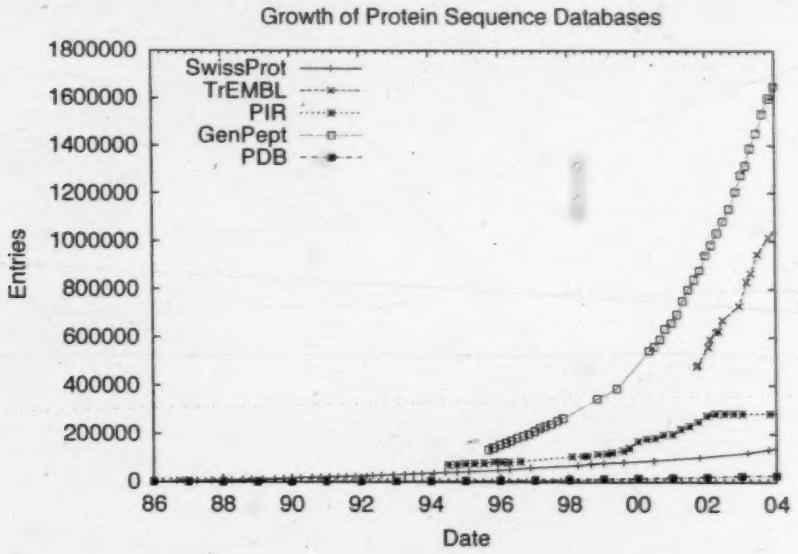
Table 1.2: The Amino Acids and the Genetic Code

The 20 amino acids used to make proteins are each indicated by a single letter code. Each amino acid corresponds to one or more triplet codons of DNA. The remaining triplets (TAA, TAG, and TGA) are 'stop' codons that signal the termination of the amino acid sequence.

Amino Acid	Symbol	Genetic Code
Alanine	A	GCT, GCC, GCA, GCG
Arginine	R	AGA, AGG, CGT, CGC, CGA, CGG
Asparagine	N	AAT, AAC
Aspartic acid (Aspartate)	D	GAT, GAC
Cystine	C	TGT, TGC
Glutamine	Q	CAA, CAG
Glutamic acid (Glutamate)	E	GAA, GAG
Glycine	G	GGT, GGC, GGA, GGG
Histidine	H	CAT, CAC
Isoleucine	I	ATT, ATC, ATA
Leucine	L	CTT, CTC, CTA, CTG, TTA, TTG
Lysine	K	AAA, AAG
Methionine	M	ATG (start)
Phenylalanine	F	TTT, TTC
Proline	P	CCT, CCC, CCA, CCG
Serine	S	AGT, AGC, TCT, TCC, TCA, TCG
Threonine	T	ACT, ACC, ACA, ACG
Tryptophan	W	TGG
Tyrosine	Y	TAT, TAC
Valine	V	GTT, GTC, GTA, GTG
Ambiguous Symbols	Symbol	
Aspartic acid or Asparagine	B	
Glutamic acid or Glutamine	Z	
Any Amino Acid	X	

Figure 1.2: Growth of Protein Sequence Databases

The SwissProt and TrEMBL databases (now UniProt [9]) have quickly increased in size since their inception in 1986. The SwissProt database is a human annotated protein database, while the TrEMBL database is an automatically annotated supplement to SwissProt.
It is clear that the human annotators cannot keep up with the huge amount of protein sequences that have been added to the TrEMBL database. Automated annotation is required to make better use of the sequence information that is available.
Data courtesy of the NIAS DNA Bank (National Institute of Agrobiological Sciences, Japan).



in protein data (see Figure 1.2). The growth levels of human-annotated databases such as SwissProt cannot keep up with the pace of TrEMBL and GenPept. For this reason, highly automated methods of sequence analysis for protein classification are needed in order to obtain useful knowledge from these mountains of data. We need automatic and high-throughput sequence analysis methods in order to reduce the impact of the human annotation bottleneck.

Machine learning involves the task of learning from data in order to predict some characteristic of interest on new data. Through the use of machine learning techniques, we can utilize the large amounts of characterized proteins that are already described in scientific databases to learn patterns that correlate with protein function. The learned patterns can then be used to predict the functions of proteins that are yet to be characterized. These machine learning techniques have been developed to allow very automated analysis, greatly reducing the human input required in the process. Many of these techniques have also been designed to be very

efficient, lending themselves to high-throughput classification tasks.

It is the goal of this work to examine the use of machine learning methods to allow accurate, automatic, and high-throughput classification of proteins from sequence data. By this, we hope to reduce the bottleneck of 'wet lab' work and human resources where scientists characterize a protein's function through a series of painstaking and expensive experiments. Where accurate predictions are possible, resources will be freed for the more interesting and difficult characterizations. By being able to make many predictions at once, we also enable investigators to view larger groups of proteins in ways they previously have not and perhaps find associations that were not previously known to exist.

We will compare current protein function prediction techniques in the context of the high-throughput and automated classification task. A recently developed probabilistic model called the probabilistic suffix tree (PST) will be further developed. The current work presents an alternative representation of PSTs, as well as results using an efficient implementation of the model. The results will show the promise of this new tool for automatic and high-throughput protein function prediction.

1.3 Protein Classes

Protein classes or families are defined by common characteristics among groups of proteins. These commonalities may be of almost any type and vary according to the research perspective of those studying the proteins in question. There are many classification systems in place and there is potential for many more.

Classification systems may differ greatly in their structure. Some are hierarchical in nature, defining groups and subgroups of classification. Others are relatively flat and define no relationship between the various classes. Some classes are mutually exclusive while others are allowed to overlap. It should also be recognized that the use of terminology within the scientific community changes between researchers and over time. Because of this, classification systems also evolve over time. Differences in the classification systems and the biological realities they reflect may impact upon the way in which class prediction is done. A small sample of these classification systems follows.

Gene Ontology The Gene Ontology [28] (GO) is a large hierarchical classification system (see Figure 1.3) that attempts to unify classification from diverse research interests. There are three hierarchies defined for Gene Ontology – Molecular Function, Cellular Process, and Cellular Component. The hierarchies include both high-level classes such as 'catalytic activity' (GO:0003824) and very specific low-level classes such as 'pre-mRNA branch point binding' (GO:0045131). We will focus mainly on the higher-level part of the Molecular Function hierarchy in this

Figure 1.3: The Gene Ontology

The first three levels of the Gene Ontology Molecular Function Hierarchy are shown. The selected sample of Gene Ontology classes used for classification in this work is shown in Figure 1.4.

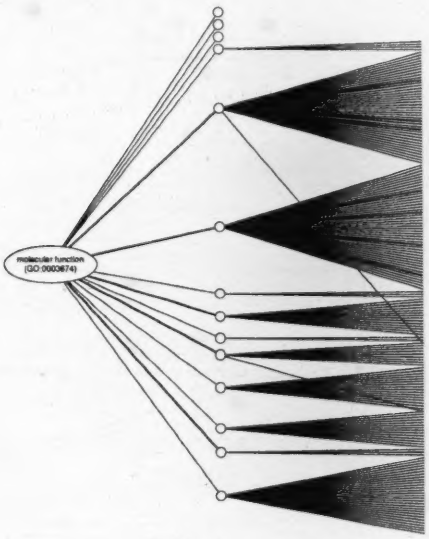
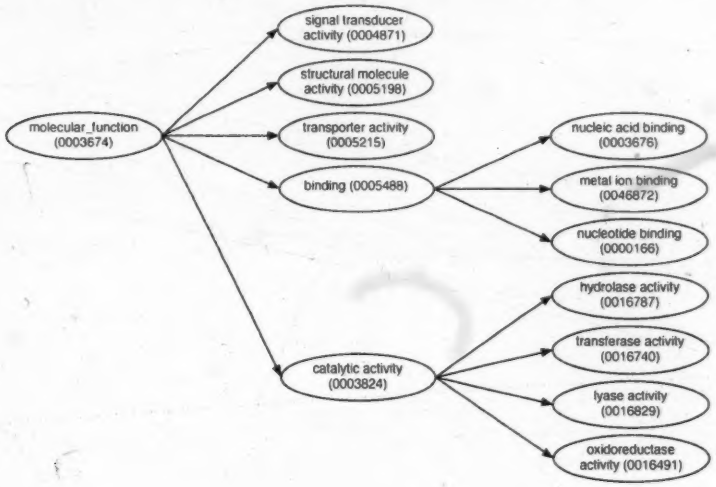


Figure 1.4: A Gene Ontology Sample

The hierarchy shows the relationships in the selected sample of Gene Ontology classes used for classification in this work. A class name is shown with the unique GO identifier in brackets.



work (see Figure 1.4), as well as some elements related to the Cellular Component hierarchy.

Subcellular Location The subcellular location of a protein is the cellular compartment in which it exists. The compartment in which a protein exists is usually determined by experimental means. The Cellular Component hierarchy of the Gene Ontology describes subcellular location.

The subcellular locations that are available differ according to the organism in which the proteins are found. Prokaryotic organisms, for example, lack the nucleus found in all eukaryotic organisms and do therefore have 'nucleus' as a possible subcellular location.

Although the definition of subcellular location is relatively straightforward, some variation in the assignment of subcellular location of specific proteins still exists. Many proteins pass through other cellular compartments on their way to the location at which they perform their function. In addition, some proteins function at the interface between two compartments or in multiple compartments. It is hoped that the conventions used for assigning subcellular location will be consistent within a classification scheme.

Figure 1.5: Voltage-Gated Potassium (K^+) Channel Structure

A diagram representing the structure of a voltage-gated potassium channel. Two potassium ions can be seen in the core of the protein. Image from the PDB [18].

Pfam The Pfam database [13] is a 'large collection of multiple sequence alignments and hidden Markov models covering many common protein domains and families'. These are a mix of both structurally and functionally defined protein families. Membership in each family in the database is determined by the use of hidden Markov models (HMMs) that are created from sequences in the database. A protein may belong to multiple families and thus match multiple HMMs[2].

SCOP The SCOP database [73] is a 'Structural Classification Of Proteins'. The goal of the database is to describe the folds, families, and super-families that exist in proteins. Recent versions contain all proteins that are found in the PDB [18] protein data bank of protein structures.

EC The enzyme classification system [60] by the Enzyme Commission (EC) is a hierarchical classification system for enzymes by their activity.

Potassium (K^+) Channel Proteins Voltage-gated potassium (K^+) channels (see Figure 1.5) are proteins that facilitate the transfer of potassium across membranes and are critical for the maintenance and regulation of membrane potential for many

[2]Each family is defined independently of the other families in the database except that no amino acid residue (at a particular location) in a protein can belong to two families.

important processes in physiology. These proteins have been divided into four classes in a dataset from Warren Gallin [59] at the University of Alberta and represent a typical small classification scheme that might be the focus of an individual research lab. The sequences are fairly well conserved within the classes and there is also much sequence similarity among the classes. This small set of proteins is useful in illustrating the methods developed in this work.

1.4 Related Work In Protein Function Prediction

1.4.1 Protein Function Determination

The determination of protein function has initially been the domain of experimental methods. These methods study protein expression, function, and activity through a variety of means. These methods include microscopic images, genetic experiments, biochemical tests, protein-protein interaction tests, subcellular location determination, function determination, NMR and mass spectrometry studies, and many others. While these techniques have historically been quite labour-intensive and slow, many automatic and high-throughput improvements have been developed. Despite these improvements, the availability of sequence data far outpaces the resources available to determine the function of proteins via these techniques.

1.4.2 Protein Function Prediction

Function prediction from sequence information is needed to supplement the laboratory work of protein function determination so that researchers can direct resources to the proteins of most interest. For example, protein function prediction can help select the most likely research targets from the huge number of proteins that have potential therapeutic uses in medicine. The selected protein candidates can then be studied using more expensive and rigorous 'wet lab' procedures.

Prediction methods seek to take advantage of the hard won results of years of biological experiments. They predict the function of proteins based on the most easily determined protein properties, allowing other resources to be directed to questions that cannot yet be answered in automated and high-throughput ways. This work will focus on data that has been obtained in the form of protein sequence information.

Each sequence analysis seeks to find regions of sequence similarity or patterns. The most common measure of similarity is by sequence alignment, but many other pattern types are used (as discussed in Chapter 3).

Two of the most significant current techniques for high-throughput protein function prediction are sequence alignment (BLAST [4]) and hidden Markov models (Pfam [13] and HMMer [35]). In addition, there are two notable function prediction

systems, InterProScan [8] and Proteome Analyst [90, 91], that incorporate multiple prediction technologies.

Sequence Alignments

The most common method of protein function prediction via sequence analysis has been through the use of sequence alignments (see Section 3.2.3). The score of a sequence alignment measures the similarity between sequences. In addition, the alignment produced shows common sequence regions and enables qualitative assessment of the similarities. Typically, researchers will find the most similar sequence in a database of annotated proteins and, if the similarity is sufficient, assume that the unannotated protein has the same function 'by similarity'. This type of nearest-neighbor prediction (see Section 2.4.1) is used extensively in both manual and automated annotations. The most commonly used alignment search program, BLAST [4], has become such an essential tool for molecular biology that it has attained the status of a verb (BLAST your sequence, for example).

Hidden Markov Models

Sequence patterns are also used to make function predictions. Patterns are often characterized as deterministic or probabilistic. One of the most commonly used types of pattern for representing sequence commonalities are probabilistic patterns called hidden Markov models (HMMs). Many HMMs have been designed for use with particular protein families. The most well-known database of these patterns is the Pfam database [13], where each protein family is represented by an HMM. These HMMs are usually more sensitive than BLAST searches and may be used to find members of protein classes that are divergent in many parts of the sequence except the most functionally important regions.

Pattern Databases

Groups of previously characterized patterns can be used for function prediction. InterProScan [8] has been developed as a tool to predict protein classification for the Gene Ontology and InterPro classification schemes by combining many pattern based prediction tools and databases into a single application. Each of the patterns corresponds to some GO or InterPro [71] class. After scanning new sequences with each pattern finding tool, classifications are assigned to the sequence according to each pattern matched.

Proteome Analyst

Proteome Analyst [90, 91] (PA) predicts protein function prediction from protein sequence information. PA allows prediction using user-specified classification schemes or predefined Gene Ontology and Subcellular Location classification schemes [63].

Each sequence that is processed by Proteome Analyst is matched (via BLAST) to the SwissProt database [19]. The SwissProt database is a very reliable source of protein annotations for a large variety of proteins spanning functional classes and organisms. The annotations of the sequences that are most similar to the sequences being processed by PA are extracted from the database. Keywords obtained from these annotations are then used by a classifier (produced by a machine learning algorithm – see Chapter 2) to predict the function of the sequence. The keywords contain a variety of functional information about the matched sequences, including information about Pfam and other pattern matches.

The Proteome Analyst system provides a publicly available, high-throughput, web-based system for predicting various properties of each protein in a given set of proteins (which could be an entire proteome – the set of all proteins in an organism). Each prediction is also accompanied by an explanation of the data and reasoning behind the prediction.

1.4.3 Limitations

The computational methods discussed above have greatly improved the state-of-the-art in automatic protein function prediction in a variety of ways.

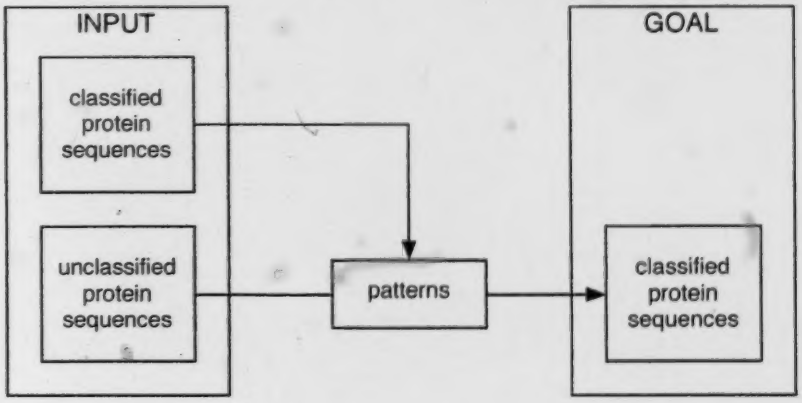
- BLAST decreased the time required for protein sequence alignment to the point that one of the first things that researchers do after obtaining a protein sequence is to quickly perform a BLAST search.
- HMMs have greatly increased the sensitivity in searching for more diverged protein domains and families.
- InterProScan incorporates a variety of useful tools and databases available for making automated predictions based on widely-used classification schemes.
- Proteome Analyst allows function prediction with a user-defined classification system and explains the rationale behind the prediction.

These techniques do not, however, address all that we might hope for in an automatic high-throughput protein classification system. The following weaknesses of these techniques limit their utility.

- *Limited Coverage:* BLAST is limited to sequences in an annotated database that have a high scoring sequence alignment (see Section 5.3.1). Proteome

Figure 1.6: The Research Goal

The goal of this research is to use the information from classified proteins to determine the classifications of new proteins. We seek to do this in a high-throughput way.



Analyst has this same limitation with the additional constraint that matches must be in the SwissProt database. Pfam and InterProScan are limited to domains and patterns that have been discovered and annotated. To this point, these patterns do not cover all the known proteins.

- *Static Classification Systems:* Pfam and InterProScan do not allow flexible definition of user-defined classification schemes. BLAST, however, does not explicitly make a prediction and is not specific to any classification scheme. As Proteome Analyst allows custom classification [90, 91], it is not limited to static pre-defined classification systems either.

- *Time/CPU Intensive:* Although BLAST and many pattern scanning tools have improved over recent years, their computational demands for high-throughput prediction are still very high. As Proteome Analyst uses BLAST, it has the same efficiency limitations that BLAST does. Although scanning with many pattern analysis tools such as Pfam is computationally expensive, the creation of the patterns is typically more expensive still.

1.5 Research Goal

The goal of this research is to develop techniques that will support automated high-throughput protein function prediction. In order for a function prediction system to work, it should have a number of characteristics.

- *Accuracy:* The system should be able to accurately predict function. The number of both false positive and false negative predictions should be minimized.
- *Efficiency:* The system should be able to predict quickly and scale up to efficiently predict function for upwards of tens of thousands of proteins.
- *Flexibility:* Because each research question is slightly different, the system should allow prediction using a user-defined classification system.
- *Transparency:* Given a prediction, the reasoning behind the prediction should be apparent to the interested user. The predictor should not be a 'black box'.

The purpose of this work is to develop sequence analysis methods that will help better achieve these goals.

A recently described and potentially very efficient probabilistic model, the probabilistic suffix tree (PST) will be evaluated as a candidate method for finding patterns in protein sequences. The potential of PSTs will be explored for use in automated and high-throughput protein function prediction. An alternate representation and implementation of PSTs will be developed that allows better realization of the above goals. We will seek to reduce the dependence on alignment-based methods to avoid coverage problems and to also reduce the dependence on less computationally efficient models such as HMMs.

It is intended that this work will be integrated into systems such as Proteome Analyst to improve both coverage and efficiency.

1.6 Summary

This work will examine prior research in protein sequence analysis in the context of high-throughput and automated protein function prediction. The utility and limitations of state-of-the-art techniques will be discussed. A promising new probabilistic model, the probabilistic suffix tree, will be discussed. A new formulation and implementation of that model will be demonstrated. Future directions for PSTs and related prediction tools will then be examined in the context of high-throughput and automatic protein function prediction.

We hope that by accelerating the classification of proteins, both computational and laboratory resources might be more efficiently utilized and that scientific progress might also be accelerated. The purpose of this thesis is to demonstrate the effectiveness of current techniques for high-throughput protein function prediction based on sequence information and to show that probabilistic suffix trees offer a method of increasing the efficiency of these predictions.

Chapter 2

Machine Learning

2.1 Introduction

Machine learning [70, 42, 53] deals with the problem of learning automatically from data. Despite concerns raised by science fiction about learning machines[1], machine learning involves the relatively innocuous task of analyzing patterns or associations in data. Those patterns may be examined in either a *supervised* or *unsupervised* manner. The goal of the supervised learning problem is to learn patterns from labeled training data in order to predict labels for previously unseen (and unlabeled) test data. The goal of the unsupervised learning problem is to group or cluster unlabeled data based on observed patterns or associations. Both supervised and unsupervised learning are used extensively in bioinformatics research. The following overview will focus on machine learning as relating to the protein function prediction problem. A variety of references cover general machine learning topics in more depth than is possible here [70, 42, 53].

[1]The Terminator [26], a machine from the year 2029, summarized some fears about learning machines.

> The Terminator:The Skynet Funding Bill is passed. The system goes on-line August 4th, 1997. Human decisions are removed from strategic defense. Skynet begins to learn at a geometric rate. It becomes self-aware at 2:14 a.m. Eastern time, August 29th. In a panic, they try to pull the plug.
>
> Sarah Connor: And Skynet fights back.

Fortunately, through the efforts of Sarah and John Connor and the protection of the Terminator, the demise of the human race is delayed for at least another movie. In addition, no known machine learning algorithms have yet developed the ability to become self-aware, nor is that a goal of machine learning research.

2.2 The Classification Problem

The problem of protein function prediction in sequences is predominantly a *classification* task[2]. Classification problems are supervised learning tasks in which the *classifier*[3] learns patterns from a labeled *training set*. A training set $T = (X, Y)$ consists of a vector of *instances* $X = \langle x^1 \dots x^n \rangle$ and a vector of corresponding class *labels* $Y = \langle y^1 \dots y^n \rangle$. For example, each instance of the training set (x^j, y^j) may correspond to an individual protein x^j and its known function y^j. Each protein instance may be represented as a vector of *features* or *attributes* $x^j = \langle x_1^j, x_2^j, \dots, x_m^j \rangle$. The learning process returns a *classifier* $f(\cdot)$ that can subsequently be used to make a label prediction $\hat{y}^j$ for each instance x^j of an unlabeled *test set*. This process is illustrated diagrammatically in Figure 2.1.

A classifier may be as simple as a threshold for a single continuous attribute where values that are higher than the threshold value indicate a positive label and values that are equal to or lower than the threshold indicate a negative label (see Figure 2.3). As there is not usually a single feature that allows *separation* of the positive and negative classes, more complicated classifier models that consider more complex relationships between features and classes are often required.

2.3 Model Selection

For each application we want to select the appropriate model for the task. We typically want the model that will give us the best *classification performance*.

2.3.1 Performance and Error

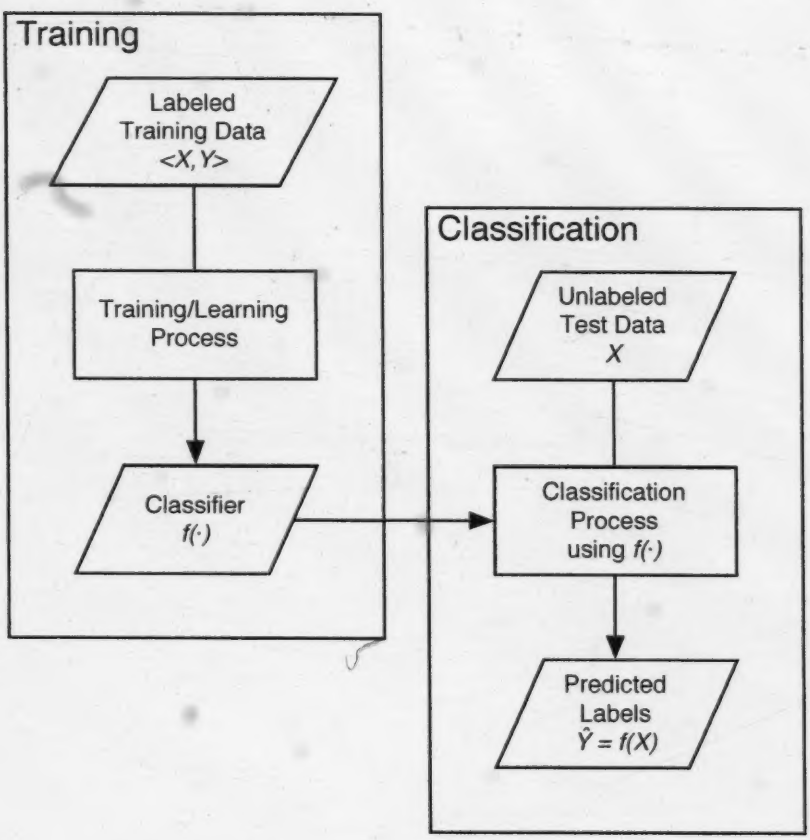
Classification performance typically means that we want to be 'right' as often as possible. In other words, we would like to select a prediction model that minimizes the expected classification error on independent test data (given that the data is drawn from the same distribution). This is the *prediction error*, also known as the *test error* or *generalization error*.

Unfortunately, we cannot really know the true prediction error beforehand because we will not have seen the true test set (in our case protein sequences that have not yet been discovered). Instead, we try to estimate the prediction error. One way to try to estimate the test error is by evaluating the *training error* (or *resubstitution error*). This is the error rate obtained by using the classifier to classify instances of the training set that was used to create it and by then comparing the predicted

[2]Classification and regression are both supervised learning problems. Classification predicts a label (qualitative, discrete), while regression predicts a value (quantitative, continuous).

[3]This and many of the terms that follow are defined in Kohavi and Provost's Machine Learning Glossary of Terms[53].

Figure 2.1: The Training and Classification Processes



labels to the known labels. Unfortunately, training error tends to be an optimistic, and thus poor, estimate of the true prediction error [42].

2.3.2 Hold-Out and Cross-Validation

To estimate the prediction error and overfitting, hold-out training data (or *validation sets*) are commonly used to simulate unknown test data. Training is performed on the training set with the exception of the hold-out data. The classifier is then used to predict labels for the hold-out set and the predicted labels are compared to the known labels in order to evaluate performance. Validation compares the results of the predicted label $\hat{y}^j$ with the known label y^j. Performance measures such as accuracy are calculated in terms of these results.

Although the use of hold-out test sets for estimating classifier performance is an attractive idea, data is typically costly and we would like to make maximal use of the data available to us. Cross-validation is a technique that allows us to get a reasonable estimate of the prediction error while using all of the data available for both training and testing.

In k-fold cross-validation, the training data is split into k folds where $k \in \{2, 3, \ldots, n\}$. There are then k iterations of testing on each fold with a classifier that has been trained on the remaining $k-1$ folds. For example, given a training set with 9 instances $(x^1, x^2, \ldots, x^9)$, a 3-fold cross-validation would first partition the set into 3 folds – say (x^1, x^4, x^7), (x^2, x^5, x^8), (x^3, x^6, x^9) – and perform three iterations of testing:

1. Train on folds 2 and 3, test on fold 1.

Testing	Training
x^1, x^4, x^7	$x^2, x^5, x^8, x^3, x^6, x^9$

2. Train on folds 1 and 3, test on fold 2.

Training	Testing	Training
x^1, x^4, x^7	x^2, x^5, x^8	x^3, x^6, x^9

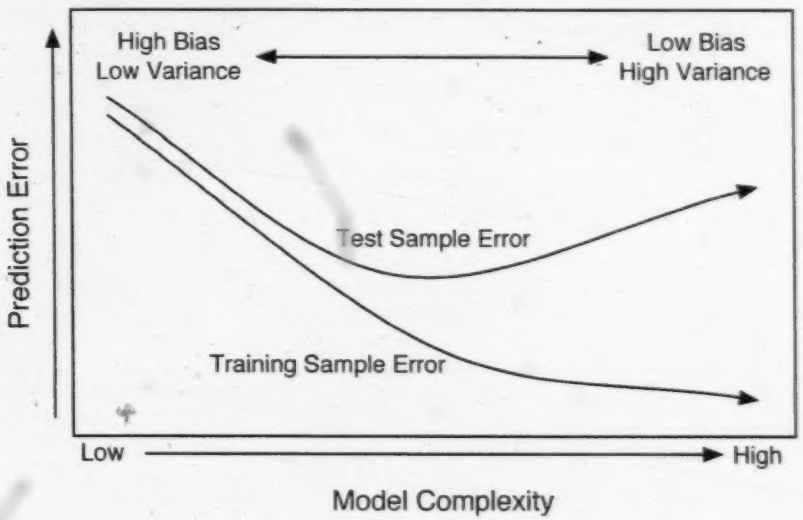
3. Train on folds 1 and 2, test on fold 3.

Training	Testing
$x^1, x^4, x^7, x^2, x^5, x^8$	x^3, x^6, x^9

The performance measures of all folds are combined to obtain overall estimates of the performance of the classifier. Typical values for k are 5 or 10. Leave-one-out cross-validation is also used (where $k = n =$ the size of the training set). In general, the size of k may be changed depending on factors such as the classifier's sensitivity to the size of the training set and the cost of each cross-validation iteration. Leave-one-out cross-validation may give a very accurate measure of the error but may

Figure 2.2: Overfitting

As model complexity increases, training error tends to decrease. Prediction error does not, however, decrease perpetually with decreasing training error. Prediction error decreases until the increasing model complexity begins to overfit the training data. This situation is also framed in the context of the bias-variance tradeoff. This figure was adapted from Hastie *et al.* [42].



also be prohibitively expensive. If the classifier is reliable, the results should be comparable for each of the k folds.

2.3.3 Overfitting

Overfitting is a typical problem in many machine learning applications. With many types of models the training error can be decreased by simply allowing the complexity of the model to increase until it can correctly classify all the training data. This scenario often results in overfitting - a situation in which the model fits the training data very well but will not generalize well to independent test data. With increasing model complexity the training error may continue to decrease but the prediction error may plateau or actually increase (see Figure 2.2). One way of explaining this effect is by the *bias-variance tradeoff*. The bias, the average amount by which the best model in the class differs from reality, tends to decrease as the model complexity increases. The variance – that is, the expected deviation of a model (learned from a data sample) from its average – tends to increase with model complexity for a fixed data sample. An increase in model complexity is worthwhile if the reduc-

tion in the (squared) bias offsets the increase in variance. See Hastie, Tibshirani and Friedman [42] for a complete discussion of this tradeoff. When the training process returns a model that closely fits the training set (low bias) but does not generalize well (high variance), overfitting is being observed. Beyond some point in training, it is possible that bias and variance cannot both be improved simultaneously – one may have to be sacrificed for the other. Overfitting[4] is of particular concern when estimating the parameters of probabilistic models.

2.3.4 Performance Measures

In general, we want to evaluate performance (and error[5]) in a way that is appropriate to the application for which the classifier is intended. For this reason, the simple performance measures of accuracy and error (equal to 1 minus the accuracy) may not always provide enough useful information about the effectiveness of our classifier. There are cases, for example, in which some errors are more costly than others. In these cases, it is not sufficient to simply reduce error. In other cases, imbalanced data sets (where there are many more instances of one class than others) may give misleading results[6]. Many performance metrics have been devised to deal with these different [53] situations.

Each of the performance metrics can weight errors to reflect their cost. In binary (two-class) classification problems, there are two types of error. Type I error is the error due to false positives. In our context this is the case where we predict that a protein has a certain function that it does not, in fact, have. Type II errors are false negatives. In this case a false negative would involve predicting that a protein does not have a certain function when, in fact, it does. These errors may have different costs. For example, if we classify a protein as being a potential target for a cancer cure and it turns out not to be successful (false positive), the cost is that of performing the experiments on that protein. On the other hand, if we classify

[4]Another view of the issue of generalization is seen through the *minimum description length principle* (MDL) and *Occam's Razor*. While the principles of MDL heuristics are discussed elsewhere [42, 70], Occam's razor [1] is a useful summary of the idea. In the 14th century a Franciscan friar named William of Ockam wrote *Pluralitas non est ponenda sine neccesitate*, which translates as 'Plurality should not be posited without necessity'. Today, the principle is often stated as 'Of two equivalent theories or explanations, all other things being equal, the simpler one is to be preferred.'

[5]For many algorithms, the concept of a *loss function* is used as a generalization of error. The loss function may be defined simply as the error rate or as a modified function of the error. Either way, the algorithm or the model selection process may be tuned to minimize loss rather than simply error. The concept of minimizing loss is similar to minimizing *cost* or maximizing *utility*.

[6]Consider a classification problem with 900 positive instances and 100 negative instances. If we used a classifier that predicted the majority class (positive) every time, we would get an accuracy of $900/1000 = 0.90 = 90\%$. Although this seems much better than the 50% we would get from a random classifier, this classifier has obviously not done much 'learning' and accuracy is a misleading indication of performance.

a protein as not being a potential target for a cancer cure and in reality it is (false negative), the cost is that of not being able to cure cancer (and, of course, missed revenues for the pharmaceutical company involved). Because false positives are much less costly than false negatives in this case, pharmaceutical companies are willing to screen many proteins for therapeutic properties in the hopes that some will be successful.

True positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) are defined in terms of the predicted label $\hat{y}^j$ and the known label y^j as shown below.

$$TP(y^j, \hat{y}^j) = \begin{cases} 1 & \text{if } y^j = true \text{ and } \hat{y}^j = true \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

$$TN(y^j, \hat{y}^j) = \begin{cases} 1 & \text{if } y^j = false \text{ and } \hat{y}^j = false \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

$$FP(y^j, \hat{y}^j) = \begin{cases} 1 & \text{if } y^j = false \text{ and } \hat{y}^j = true \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

$$FN(y^j, \hat{y}^j) = \begin{cases} 1 & \text{if } y^j = true \text{ and } \hat{y}^j = false \\ 0 & \text{otherwise} \end{cases} \tag{2.4}$$

$$\tag{2.5}$$

When the sum is taken over all the training instances, these measures are used to define various measures of classifier performance.

$$TP = \sum_{j \in Instances} TP(y^j, \hat{y}^j) \tag{2.6}$$

$$TN = \sum_{j \in Instances} TN(y^j, \hat{y}^j) \tag{2.7}$$

$$FP = \sum_{j \in Instances} FP(y^j, \hat{y}^j) \tag{2.8}$$

$$FN = \sum_{j \in Instances} FN(y^j, \hat{y}^j) \tag{2.9}$$

The errors, along with true positives and true negatives, may be displayed in a table called a *confusion matrix* (see Table 2.1 and Figure 2.3).

The following performance measures have been used for a variety of situations in machine learning, data mining, information retrieval, medicine, and statistics [53].[7]

[7]It is also possible to weight these performance measures according to the importance of particular classes or instances for a cost-sensitive evaluation measure. The only cost-related parameter considered here will be the one used in the F-measure to prioritize precision or recall.

Figure 2.3: Confusion in a Diagnostic Test

Diagnostic tests often give a numeric value as a result. A classifier may define a threshold such that values above the threshold are considered positive and values below the threshold are considered negative. The top chart shows data that is separable by a threshold given the test value. The bottom chart shows data that is not separable given the test value. In this case, false positives and/or false negatives are inevitable for a single threshold value. As the threshold is varied the true positive and true negative rates vary, leading to tradeoffs such as those observed between precision and recall or sensitivity and specificity.

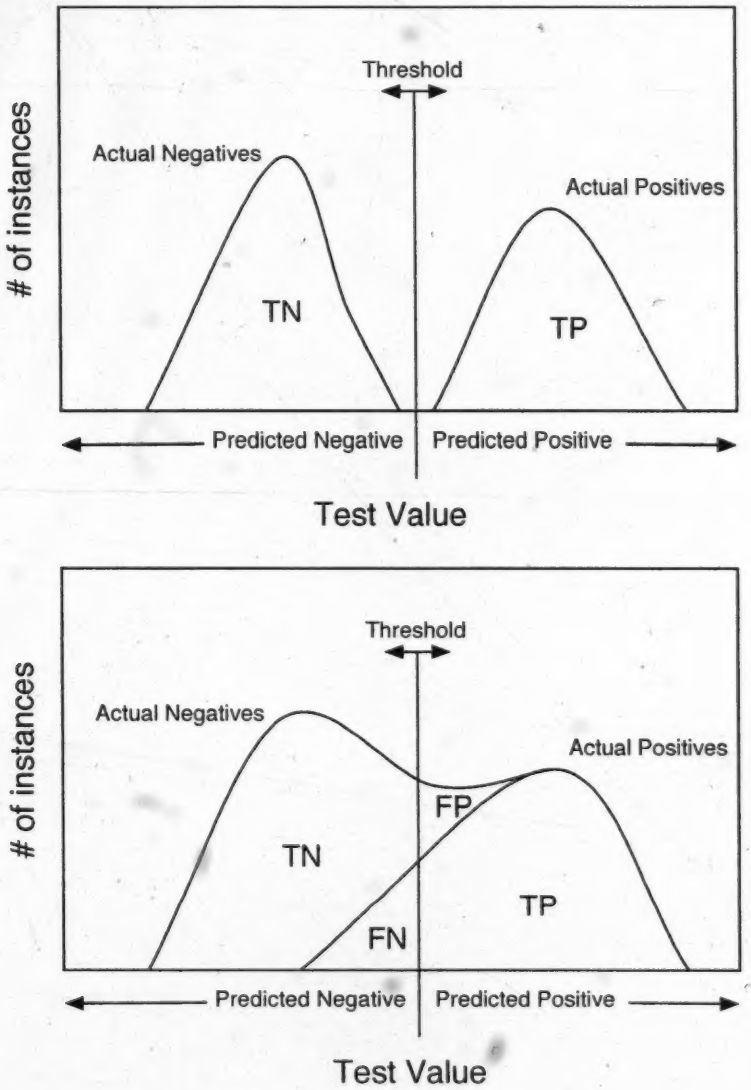


Table 2.1: Confusion Matrix

This is a simple 2x2 confusion matrix that results from the prediction of a binary classification. Confusion matrices can also be extended to use with multiple (usually mutually exclusive) classes.

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Accuracy This is the most common measure of performance in which both FP and FN are weighted equally. Accuracy is often used when a single number is desired to assess performance.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.10}$$

Precision Of those instances that are predicted to be positive, precision shows the fraction that are actually positive.

$$Precision = \frac{TP}{TP + FP} \tag{2.11}$$

Recall (Sensitivity or True Positive Rate) Of those instances that are actually positive, recall shows the fraction that are predicted to be positive.

$$Recall = \frac{TP}{TP + FN} \tag{2.12}$$

Specificity (True Negative Rate) Of those instances that are actually negative, specificity shows the fraction that are predicted to be negative.

$$Specificity = \frac{TN}{TN + FP} \tag{2.13}$$

False Positive Rate Of those instances that are actually negative, the false positive rate is the fraction that are predicted to be positive.

$$FalsePositiveRate = \frac{FP}{TN + FP} \tag{2.14}$$

False Negative Rate Of those instances that are actually positive, the false negative rate is the fractions that are predicted to be negative.

$$FalseNegativeRate = \frac{FN}{TP + FN} \tag{2.15}$$

F-measure To acknowledge the tradeoff between precision and recall and the difference in cost between false positives and false negative, the F-measure [50] has been used in information retrieval as a combined measure of performance. The weight placed on precision or recall varies with the value of the parameter β(see Equation 2.16). When $\beta = 1$, precision and recall are weighted equally. Recall is favored when $\beta < 1$ and precision is favored when $\beta > 1$.

$$Fmeasure = \frac{(\beta^2 + 1) \times Precision \times Recall}{\beta^2 \times Precision + Recall} \tag{2.16}$$

Receiver Operating Characteristic (ROC) curves ROC curves [99, 3] are commonly used to display the tradeoff between sensitivity and specificity for a variety of classification parameters. They were first used to evaluate electronic signal detection and are now often used in medical diagnosis [64] and many other fields. The false positive rate (1-specificity) is plotted against sensitivity (see Figure 2.4). A curve is obtained by plotting the false positive rates and sensitivities for a variety of parameter values. The varying parameter may actually be a parameter for the learning process or a threshold on a continuous result value. The area under the resulting ROC curve (AUC) may be used as a measure of the overall performance of the classifier. Greater area under the curve indicates a more discriminating classifier.

Sensitivity/Specificity Plots The tradeoff between sensitivity and specificity is often plotted against certain changing model parameters in order to allow intelligent parameter selection.

Multiple Classes The evaluation of performance used for binary classification tasks can be generalized to classification using multiple classes. There are two cases of multi-class classification. The classes are either 1) mutually exclusive or 2) overlapping (an instance may be assigned more than one class label). Some classification techniques are able to handle multiple (usually mutually exclusive) classes while some are only able to handle binary classification.

For mutually exclusive classes, a confusion matrix (see Table 2.2) allows misclassification rates and trends to be observed and analyzed.

Figure 2.4: ROC Curves

An example of a Receiving Operator Characteristic (ROC) curve. The curve shows the tradeoff between classifier sensitivity and specificity. The Area Under the Curve (AUC) is a measure of overall classifier performance. A diagonal curve from the bottom left corner of the plot to the top right shows the performance of a random classifier. The curves of other classifiers are often compared to (and should be above) this curve.

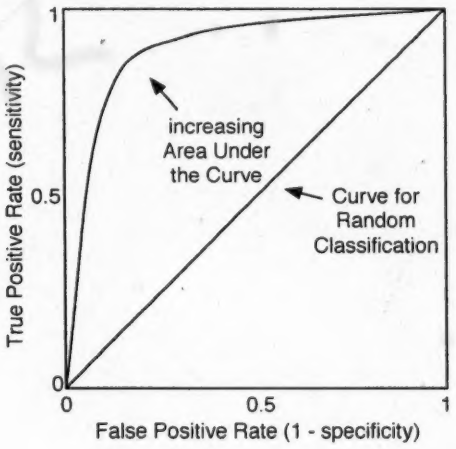


Table 2.2: Multi-class Confusion Matrix

Multiple classes that are mutually exclusive can be displayed in a confusion matrix. In the example, it can be seen that the classifier often misclassifies class A as either B or C. Classes B and C are rarely misclassified. These errors decrease the recall of prediction on class A and decrease the precision of prediction on classes B and C.

	Predicted A	Predicted B	Predicted C
Actual A	10	5	5
Actual B	0	20	2
Actual C	3	0	15

For the classification tasks considered in this work, the classes will not be considered mutually exclusive and each class will be predicted independently (ie. a single instance x^j may have multiple labels that we represent as a vector of binary values $y^j = \langle y_1^j, y_2^j, \ldots, y_m^j \rangle$ where m is the total number of possible class labels. In this case, the information presented in a confusion matrix becomes 'confusing', so other ways to present overall performance may be used.

For the problems considered in this work, we will consider both class-wise and overall performance. Predictions are evaluated as binary yes/no predictions $\hat{y}$ for each class label c on each instance (x^j, y^j). The class-wise results (TP, TN, FP, FN) and performance measures (precision, recall, etc.) are defined for each class as for a binary classification (above). The overall results are calculated as the sum of the results over all the classes.

$$TP_{overall} = \sum_{c \in Classes} TP_c \tag{2.17}$$

$$TN_{overall} = \sum_{c \in Classes} TN_c \tag{2.18}$$

$$FP_{overall} = \sum_{c \in Classes} FP_c \tag{2.19}$$

$$FN_{overall} = \sum_{c \in Classes} FN_c \tag{2.20}$$

The overall performance measures are then calculated as defined above (see Equations 2.10 to 2.16).

Coverage A classifier may opt to return no prediction when there is insufficient evidence. In such circumstances the coverage may be defined to be the number of predictions made out of the total number of predictions requested of the classifier. When performance measures are calculated the number of 'no prediction's may be either incorporated or ignored (since they are already considered when reporting coverage). In the present work, results without predictions will be incorporated into the performance measures. For this, 'no prediction' results that are known to be positive (NPP) will be differentiated from 'no prediction' results that are known to be negative (NPN). This leads to the redefinition of accuracy and recall (precision is not affected).

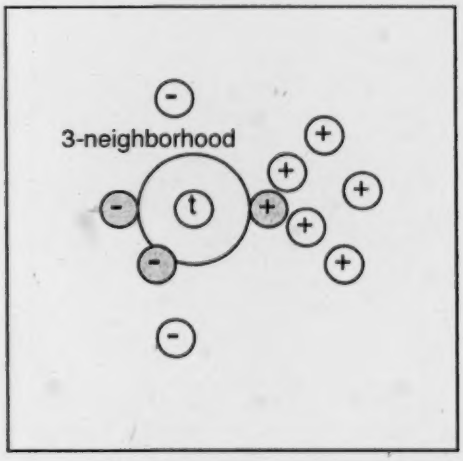
$$Coverage = \frac{TP + TN + FP + FN}{TP + TN + FP + FN + NPP + NPN} \tag{2.21}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN + NPP + NPN} \tag{2.22}$$

$$Recall = \frac{TP}{TP + FN + NPP} \tag{2.23}$$

Figure 2.5: Nearest-Neighbor Classification

The predicted class of test instance t is negative – the majority class of the 3 nearest neighbors (shown in gray).



2.4 Examples of Classification Techniques

Many classification techniques have been used in machine learning. Each has inherent advantages and disadvantages given the nature of the features and patterns of features in the data set. Classifiers may vary in their accuracy, robustness, or time and memory requirements. The examples below are a small sample of the classification techniques available.

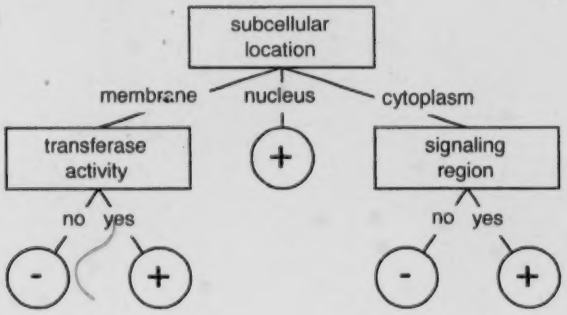
One should be aware that preprocessing techniques, including the filtering, combining, and discretizing of features may be crucial to the improvement of classifier performance. Exploration tools such as WEKA [98] allow users to experiment with a wide range of techniques and select the one that performs best for the target data set.

2.4.1 Nearest-Neighbor

Nearest-neighbor classification classifies a test instance based on its 'neighborhood', or the nearest instances from the training set. The concept of 'nearness' (or similarity) requires a metric that defines the distance between two instances. For example, if the features are in Euclidean space then the metric may be the Euclidean distance. In order to increase the robustness of the method to noisy data, the k-neighborhood is often considered. Here, the average of the k nearest neighbors is

Figure 2.6: A Decision Tree

The following decision tree represents the concept of positive or negative for signaling function based on the fictional data from Table 2.3.



used to label the test instance (see an example of 3-nearest neighbor classification in Figure 2.5).

A common way to measure the distance between two protein sequences is by the BLAST alignment score [4] (see Section 3.2.3). Smaller BLAST E-value scores indicate decreasing distance and increasing similarity between sequences. The neighborhood of a test sequence may be obtained by performing BLAST alignments against each sequence in the training set. If the majority of the neighbors in the k-neighborhood of the test protein has a particular class label then the test protein is predicted to also have that label. Molecular biologists use BLAST scores and databases routinely on new protein sequences to identify potential functions. As such, a nearest-neighbor BLAST classifier may be considered the default classification technique for biological sequences.

The time requirements of nearest-neighbor classification depend mainly on the time required to evaluate the distance measure. Although nearest-neighbor classification may often have high accuracy and be relatively intuitive, it does not provide a single model afterward that can be interpreted as a pattern. The pattern is as complex as the training data because the pattern *is* the training data.

2.4.2 Decision Trees

A decision tree [79] allows classification of test instances by following a series of decisions from the root to a leaf of the tree. The choice of which branch to follow at each decision node is made on the basis of the the value of an instance attribute. The classification returned for an instance is found as the decision process reaches a leaf node containing the appropriate classification label.

Table 2.3: Example Machine Learning Data

The table contains fictional data for a protein function prediction task. In this fictitious example, the features will be used to predict whether or not a certain protein has signaling function.

Protein ID	Subcellular Location	% Proline	Transferase Activity	Potential Signal Region	Signaling Function
1	membrane	high	no	yes	-
2	membrane	high	no	no	-
3	nucleus	high	no	yes	+
4	cytoplasm	med	no	yes	+
5	cytoplasm	low	yes	yes	+
6	cytoplasm	low	yes	no	-
7	nucleus	low	yes	no	+
8	membrane	med	no	yes	-
9	membrane	low	yes	yes	+
10	cytoplasm	med	yes	yes	+
11	membrane	med	yes	no	+
12	nucleus	med	no	no	+
13	nucleus	high	yes	yes	+
14	cytoplasm	med	no	no	-

Consider the following example (see Figure 2.6), adapted for protein function classification from examples by Quinlan [79] and Mitchell [70]. The training data in this fictional example (seen in Table 2.3) can be used to build the decision tree that can then be used to predict whether or not a protein will have 'Signaling Function'. The features are known characteristics of the protein and the class label is either positive or negative for signaling function. The features in this example resemble the words (extracted from the protein annotations of similar sequences) that might be used for classification by Proteome Analyst [90, 91].

By following the tree from the root to the leaves, a protein x that has the features ('Subcellular Location' = 'membrane','% Proline' = 'high', 'Transferase Activity' = 'no', 'Potential Signal Region' = 'no') would be classified as negative ($\hat{y} = -$) for 'Signaling Function' by considering decisions along the left side of the tree. The features '% Proline' and 'Potential Signal Region' would not affect the classification. In fact, we can see from this tree that the '% Proline' feature is not be used in any classifications concerning 'Signaling Function'.

A decision tree is built by selecting features upon which to base the next decision node in a greedy manner. The feature that has the next highest score according to some *splitting criteria* is chosen as for the next node. A common splitting criteria is based on *information gain*. Based on the principle of *entropy* from information theory, the feature that gives the greatest information gain is the one that most reduces the remaining uncertainty about the classification. Given a training set T and a set of possible $Classes$ (+ and −) and $Features$ ('Subcellular Location', '% Proline', 'Transferase Activity', and 'Potential Signal Region'), $V(Feature)$ is the set of values for the given $Feature$ (for example, $V(Subcell) = \{+, -\}$) and T_v is the portion of the training set that has the value v. Information gain is defined in terms of entropy below. Note that when $p_c(T) = 0$ the value of $0 \log 0$ is defined to be 0. Thus, when all the members of a set belong to the same class the entropy will be 0.

$$Gain(T, Feature) = Entropy(T) - \sum_{v \in V(Feature)} \frac{|T_v|}{|T|} Entropy(T_v) \quad (2.24)$$

where

$$I(condition) = \begin{cases} 1 & \text{if } condition = true \\ 0 & \text{otherwise} \end{cases} \quad (2.25)$$

$$p_c(T) = \frac{\sum_{j \in T} I(y^j = c)}{\sum_{j \in T} 1} \quad (2.26)$$

$$Entropy(T) = \sum_{c \in Classes} -p_c(T) \log_2(p_c(T)) \quad (2.27)$$

The information gain can be calculated and compared for each of the features

so that the feature with the highest information gain in the training data (see Table 2.3) can be used as the top node in the (sub)tree.

$$
\begin{aligned}
V(Pot.SignalRegion) &= \{yes, no\} \\
T &= [9+, 5-] \\
T_{Pot.Sig.Reg.=yes} &= [6+, 2-] \\
T_{Pot.Sig.Reg.=no} &= [3+, 3-] \\
Gain(T, Pot.SignalRegion) &= Entropy(T) - \sum_{v \in \{yes,no\}} \frac{|T_v|}{|T|} Entropy(T_v) \\
&= Entropy(T) - (8/14) Entropy(T_{yes}) \\
&\quad -(6/14) Entropy(T_{no}) \\
&= 0.940 - (8/14)0.811 - (6/14)1.00 \\
&= 0.048
\end{aligned}
$$

$$
\begin{aligned}
V(TransferaseActivity) &= \{yes, no\} \\
T &= [9+, 5-] \\
T_{Trans.Act.=yes} &= [6+, 1-] \\
T_{Trans.Act.=no} &= [3+, 4-] \\
Gain(T, TransferaseActivity) &= Entropy(T) - \sum_{v \in \{yes,no\}} \frac{|T_v|}{|T|} Entropy(T_v) \\
&= Entropy(T) - (7/14) Entropy(T_{yes}) \\
&\quad -(7/14) Entropy(T_{no}) \\
&= 0.940 - (7/14)0.592 - (7/14)0.985 \\
&= 0.151
\end{aligned}
$$

In this example, 'Transferase Activity' gives more information gain than 'Potential Signal Region' and would thus be selected first (out of these two features) to represent a node in the tree. The tree is continually grown from the root until either there are no more features upon which the tree can branch or some other stopping criteria is reached.

There are many other issues to consider in building decision trees such as splitting, stopping, and pruning criteria. Other references contain more complete discussions of these issues [70].

In practice, decision trees provide very transparent and easily interpretable predictions. They do suffer from the inability to represent many relationships between

the features and the class (once again, see Mitchell [70] and other texts for a discussion of this). Decision trees are also relatively inefficient to train. Scalability problems prevent us from using decision trees in high-throughput tasks unless the feature space is sufficiently small or the classifier performance is significantly better than other techniques.

2.4.3 Naïve Bayes

The naïve Bayes classifier predicts the most likely class label given the features. In using a naïve Bayes classifier we make the assumption that, given the class, each feature is independent of all others. Because of this assumption, the naïve Bayes predictions can be calculated in the following way for classes C and features $\langle x_1, x_2, \ldots, x_n \rangle$ of any instance x:

$$
\begin{aligned}
PredictedClass(x) &= argmax_{c \in C} P(c|x_1, x_2, \ldots, x_n) && (2.28)\\
&= argmax_{c \in C} \frac{P(c)P(x_1, x_2, \ldots, x_n|c)}{P(x_1, x_2, \ldots, x_n)} && (2.29)\\
&= argmax_{c \in C} P(c)P(x_1, x_2, \ldots, x_n|c) && (2.30)\\
&= argmax_{c \in C} P(c)P(x_1|c) \ldots P(x_n|c) && (2.31)\\
&= argmax_{c \in C} P(c) \prod_i P(x_i|c) && (2.32)
\end{aligned}
$$

The independence assumption is key in moving from Equation 2.30 to Equation 2.31.

Consider the example sequences from Table 2.3 for the example in Section 2.4.2. Given a new protein x where $x =$ ('Subcellular Location' = 'membrane', '% Proline' = 'low', 'Transferase Activity' = 'no', 'Potential Signal Region' = 'no'), a naïve Bayes classifier that is first trained on the example data would predict

$$
\begin{aligned}
PredictedClass(x) &= argmax_{c \in C} P(c) \prod_i P(x_i|c)\\
&= argmax_{c \in C} P(c)P(Subcell.Loc. = mem.|c)\\
&\quad \cdot P(\%Pro. = low|c)P(Trans.Act. = no|c)\\
&\quad \cdot P(Pot.SignalRegion = no|c)
\end{aligned}
$$

We can calculate the probabilities of each class from counting the occurrences in the training data.

$$
\begin{aligned}
P(SignallingFunction = yes) &= 9/14 \approx 0.64\\
P(SignallingFunction = no) &= 5/14 \approx 0.36
\end{aligned}
$$

We can also estimate the conditional probabilities by counting occurrences.

$$\begin{aligned} P(Pot.SignalRegion = no|SignallingFunction = yes) &= 3/9 = 0.33 \\ P(Pot.SignalRegion = no|SignallingFunction = no) &= 3/5 = 0.60 \\ &\vdots \end{aligned}$$

These values allow us calculate the probabilities for each class given the feature vector. First, we compute

$$\begin{aligned} & P(SF = yes)P(SL = mem.|SF = yes) \\ \cdot & P(\%Pro = low|SF = yes)P(TA = no|SF = yes) \\ & \cdot P(PSR = no|SF = yes) = 0.0053 \\ & P(SF = no)P(SL = mem.|SF = no) \\ \cdot & P(\%Pro = low|SF = no)P(TA = no|SF = no) \\ & \cdot P(PSR = no|SF = no) = 0.0206 \end{aligned}$$

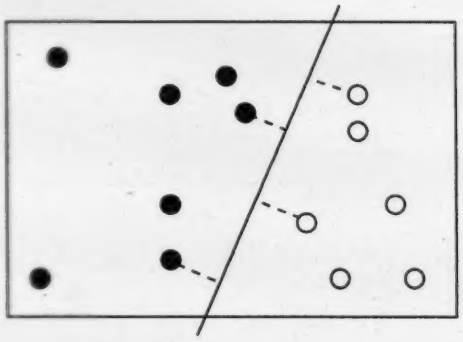
Thus, a new protein where ('Subcellular Location' = 'membrane', '% Proline' = 'low', 'Transferase Activity' = 'no', 'Potential Signal Region' = 'no') would be classified as not having signaling function. In addition, we can determine the conditional probability of the prediction by normalizing $(0.0206/(0.0206 + 0.0053) = 0.795)$.

This technology is used as the basis for the classifier of the Proteome Analyst [90, 91] protein classification system. In the case of Proteome Analyst, the features for each protein are keywords describing protein characteristics. Each keyword is either present or absent for each instance. The keywords are obtained from the database annotations of the nearest protein neighbors of the protein instance to be classified as determined by the BLAST algorithm. Once the keywords have been obtained, the naïve Bayes learning algorithm is used to create a classifier and predict the class labels of subsequent proteins.

The naïve Bayes algorithm performs surprisingly well despite using the strong independence assumption (which is typically wrong) and so it is often used in practice. Given reasonable classification performance, one of the next most attractive aspects of naïve Bayes classification is its efficiency and scalability. In addition, the classification can be easily interpreted in terms of the probabilities used by the model. Naïve Bayes classification is a popular technology for these reasons and it has been used in many applications such as spam filtering and, more importantly for our goal, Proteome Analyst.

Figure 2.7: A Separating Hyperplane

The hyperplane shown separates the two classes. The distance (or margins) from the separating hyperplane to the support vectors is maximized.



2.4.4 Support Vector Machines

Support vector machines [42, 87] (SVMs) are state-of-the-art classifiers. A good tutorial on this challenging topic has been given by Cristianini [30] and some of that material will be echoed here.

SVMs optimize the separation of labeled classes by constructing an optimal decision boundary between them. This decision boundary is learned from labeled training instances and can be applied to classify new test instances. Given an instance with a label y^j and feature vector $\vec{x}^j$, classification is based upon the hyperplane defined by the weight vector $\vec{w} \in \Re^n$ and the offset $b \in \Re$. The prediction $\hat{y}^j$ is given[8] by

$$\hat{y}^j = sign(\langle \vec{w}, \vec{x}^j \rangle + b) \tag{2.33}$$

The training process finds the optimal hyperplane that maximizes the margin between the classes (see Figure 2.7). As an aside, the solution for the weight vector $\vec{w}$, where α^j is a weight assigned to each of the training points ($\alpha^j \geq 0$), is given (without loss of generality) by

$$\vec{w} = \sum_i \alpha^j y^j \vec{x}^j \tag{2.34}$$

[8] $\langle \vec{a}, \vec{b} \rangle$ is the inner-product of the vectors $\vec{a}$ and $\vec{b}$, defined as $\langle \vec{a}, \vec{b} \rangle = \sum_i \vec{a}_i \vec{b}_i$.

Table 2.4: Example SVM Data

The table contains a mapping of the fictional data from Table 2.3. In this fictitious example, the features will be used to predict whether or not a certain protein has signaling function. The data have been converted from multi-value attributes to binary attributes for use with an SVM.

Protein ID	Subcellular Location			% Proline			Transferase Activity		Pot. Sig. Region		Signal Function
	mem	nuc	cyt	high	med	low	no	yes	yes	no	
1	1	0	0	1	0	0	1	0	1	0	0
2	1	0	0	1	0	0	1	0	0	1	0
3	0	1	0	1	0	0	1	0	1	0	1
4	0	0	1	0	1	0	1	0	1	0	1
5	0	0	1	0	0	1	0	1	1	0	1
6	0	0	1	0	0	1	0	1	0	1	0
7	0	1	0	0	0	1	0	1	0	1	1
8	1	0	0	0	1	0	1	0	1	0	0
9	1	0	0	0	0	1	0	1	1	0	1
10	0	0	1	0	1	0	0	1	1	0	1
11	1	0	0	0	1	0	0	1	0	1	1
12	0	1	0	0	1	0	1	0	0	1	1
13	0	1	0	1	0	0	0	1	1	0	1
14	0	0	1	0	1	0	1	0	0	1	0

Those instances j with weights α^j having non-zero values are informative and define the *support vectors* (since the prediction is expressed in terms of, or supported by those instances).

For our example, we might find that we get the following values[9] (for the calculated values $\vec{w}$ and b, not for the weights α^j).

$$\vec{w} = \begin{bmatrix} -0.5955 \\ 1.1905 \\ -0.595 \\ -0.3572 \\ 0.5477 \\ -0.1905 \\ -0.8095 \\ 0.8095 \\ 0.4523 \\ -0.4523 \end{bmatrix}$$

$$b = 0.5239$$

To compute the prediction $\hat{y}$, this can be written as

$$\hat{y} = sign \begin{pmatrix} & -0.5955 & * & SubcellLoc_{mem} \\ + & 1.1905 & * & SubcellLoc_{nuc} \\ + & -0.595 & * & SubcellLoc_{cyt} \\ + & -0.3572 & * & Proline_{high} \\ + & 0.5477 & * & Proline_{med} \\ + & -0.1905 & * & Proline_{low} \\ + & -0.8095 & * & Transferase_{no} \\ + & 0.8095 & * & Transferase_{yes} \\ + & 0.4523 & * & SignalRegion_{yes} \\ + & -0.4523 & * & SignalRegion_{no} \\ + & 0.5239 & & \end{pmatrix}$$

Thus, a new protein where x =('Subcellular Location' = 'membrane', '% Proline' = 'low', 'Transferase Activity' = 'no', 'Potential Signal Region' = 'no') would be classified as negative using the calculation below. The binary values are grouped by feature for clarity, but each digit is a separate value.

$$\begin{aligned} \hat{y} &= sign(\langle \vec{w}, [100\ 001\ 10\ 01] \rangle + b) \\ &= sign(-0.5995 + -0.1905 + -0.8095 + -0.4523 + 0.5239) \\ &= sign(-1.5279) = (-) \end{aligned}$$

[9]Actual values obtained from the SMO implementation of SVMs in WEKA [98] using the test data in Table 2.4.

One of the surprising and useful characteristics of support vector machines is their dual representation. The prediction can be expressed in two ways. Given the previous definition of $\vec{w}$ above we see that

$$\hat{y}^j = sign(\langle \vec{w}, \vec{x}^j \rangle + b) \quad (2.35)$$

$$= sign(\sum_i \alpha^j y^j \langle \vec{x}^j, X \rangle + b) \quad (2.36)$$

This means that we no longer require the original set of features $\vec{x}^j$ for every instance j in order to perform training and classification. We only need the dot product $\langle \vec{x}^j, \vec{x}^j \rangle$ for all instances $i, j \in |X|$. We can also (as with most machine learning algorithms) use some function of the features $\phi(\vec{x}^j)$.

$$\hat{y}^j = sign(\langle \vec{w}, \vec{x}^j \rangle + b) \quad (2.37)$$

$$= sign(\sum^j \alpha^j y^j \langle \phi(\vec{x}^j), \phi(X) \rangle + b) \quad (2.38)$$

Since the dot product $\langle \phi(\vec{x}^j), \phi(X) \rangle$ is not dependent on anything else, we can define a kernel $K(\cdot, \cdot)$ such that

$$K(\vec{x}^j, \vec{x}^*) = \langle \phi(\vec{x}^j), \phi(\vec{x}^*) \rangle \quad (2.39)$$

Over the data set X the kernel function K defines a $|X|$ by $|X|$ kernel matrix (or gram matrix). The kernel matrix (which must be symmetric positive definite) contains all the information that is needed for the learning algorithm.

The kernel function has great implications for the modularity and efficiency of SVMs (for a full discussion of the implications see the references above). The kernel function can allow the use of SVMs to classify data that may not have the typical structure of features $\vec{x} = \langle x_1, \ldots, x_n \rangle$. This enables the definition, for example, of a kernel function that might be defined for protein sequences. In this way we can take advantage of SVM technology for the protein function prediction task.

Two examples of kernels that have been defined for sequence analysis include the Fisher kernel for HMMs [47] and mismatch string kernels for sequences [57].

Chapter 3

Sequence Patterns

Alfred Whitehead said, '*Art* is the imposing of a pattern on experience, and our aesthetic enjoyment is recognition of the pattern.'[1] In the current context, 'the imposing of a pattern on experience' is also *science*. The patterns we will deal with occur in protein sequences and are learned from the experience of characterized proteins. 'The recognition of the pattern' enables us to predict protein function.

3.1 Sequence Patterns and Protein Function

To the extent that protein sequence determines protein structure and function, sequence patterns may be used to predict protein function.

Two tasks must be successfully completed in order to make use of sequence patterns. We must be able to

1. learn patterns that are correlated with a particular function and
2. recognize those patterns in new sequences.

With the increasing size of the protein databases it is not only crucial that we be able to accomplish these tasks, but that we accomplish them efficiently. This is particularly important for high-throughput automatic protein function prediction.

The efficiency of these tasks can vary widely between pattern types and also depends on both the size and complexity of the pattern. The size of sequence patterns may vary from small signal motifs of only a few amino acids in length to large structural domains that dominate the entire length of a sequence. The complexity of these patterns may vary from simple consensus sequences to complex groups of interdependent sub-patterns.

Many computational tools have been developed for the analysis of sequence patterns. Each technique is an effort to model or approximate the 'true' pattern

[1] Alfred North Whitehead, British philosopher, in *Dialogues* on June 10, 1943.

Figure 3.1: Example Protein Sequences

These fictitious protein sequence fragments are shorter than the typical protein sequences which can be from about tens to thousands of amino acids in length. There are five sequences in the example set. Each sequence is preceded by a single tag line that starts with a '>' character and contains identifying information about the sequence. All lines that do not have the '>' symbol contain sequence information. These sequences will be used in examples throughout the chapter.

```
>sequence 1
QIKDLLVSSSTDLDTTLVLRENVATLPAEKMKPFFINDAF
THEKWPATTERNSFHVTILELKYFQESKPVMPQMMCNS
>sequence 2
RRVKVYLPQMKIEEKYNLTSVLMALGMTDLFIPSANLTFI
NDMFTHEEDPATTERNSKISQAGSSAESLIGVIEDIKHSP
>sequence 3
ISEEYISYGGEKKILAIQGALEKALRWASGESFIELSNHK
FDRMFINDRKTHEKLPATTERNSSAKFRRFT
>sequence 4
AKLAEQAERYDDNLLSVAYKNVVGARRSSWRVISSIEQKT
ERNEKKQQMGKEYREKIEAELQDICNDVLELLDKFINDMK
THEKLPATTERNSYLIPNRSQPESKVFYLKMKGDYFRYLS
EVASGDNKQTVSNSQQAYQEAFEISKKEMQPT
>sequence 5
MITILEKISAIESEMARTQKNKATSAHLGGGGTGEAGFEV
AKTGDARVGFVGFPSVGKSTLLSNLAGVFINDANTHELRP
ATTERNSYSEVAAYEFTTLTTVPGCIKYKGAKIQLLDLPG
IIEGAKDGKGRGRQVIAVARTC
```

that confers the biological structure or function in which we are interested. As George Box [20] asserted, 'All models are wrong but some are useful.' By necessity, protein sequence models abstract away many of the details relating to the actual expression, structure, and function of a protein. By using only protein sequences we also ignore other details that might otherwise be available to us. For example, protein sequences do not have any inherent encoding of the placement of amino acids in three-dimensional space[2]. The DNA sequence that encodes the template for the protein sequence supplies much more potential information about the expression and genetic background of the protein in question. In both of these cases, taking into account this additional information makes the pattern recognition task much more difficult. As more information is added, more noise is added. Simplifications often allow us to both ignore superfluous information and create efficient (and potentially more 'useful') solutions. Doolittle (as quoted by Gusfield [40], pg. 390) has gone so far as to say,

> Translate those DNA sequences! Some beginning sequence comparers are under the impression there is more to be gained by searching the actual DNA sequence rather than the amino acid sequence derived from it. Such a course is greatly mistaken...

Although many of the techniques discussed here are useful for or may have originated from DNA sequence analysis techniques, this discussion will focus on protein sequences.

For most sequence analysis tasks each protein is represented as a simple sequence of symbols (representing amino acids). Each of the 20 amino acids used in proteins is represented by a single letter (see Figure 1.2), and they are collectively organized in strings that vary from tens or hundreds to thousands of amino acids in length. These strings are usually stored as text, as in the commonly used FASTA sequence file format (seen in Figure 3.1). Patterns in the protein sequences are then represented simply as patterns in the strings of symbols.

A typical example of the relationship between a sequence pattern and protein function is the zinc finger (see Figure 3.2). The zinc finger is a DNA-binding structure that is found in many proteins. The amino acid sequence that becomes the zinc finger structure is well conserved among these proteins (Figure 3.3 shows an example sequence with annotated structure that corresponds to Figure 3.2). This sequence pattern – which matches many proteins that have the zinc finger structure – can be expressed by the following regular expression[3] (pattern PS00028 from the PROSITE database).

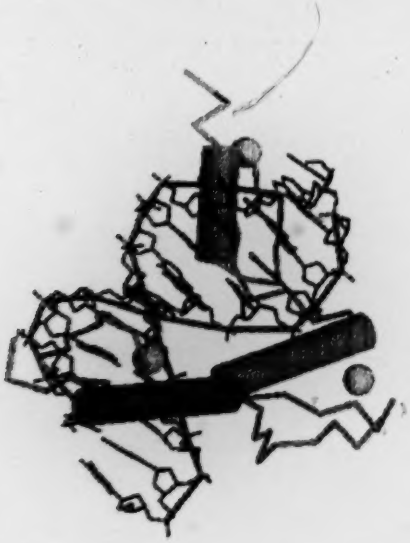
---

[2]Although primary sequence does lead to 3D structure, current technology does not allow us to easily predict structure given only sequence information.

[3]This regular expression corresponds to the following pattern: C, 2 to 4 positions of any amino acids, C, 3 positions of any amino acids, any of LIVMFYWC, 8 positions of any amino acids, H, 3 to 5 positions of any amino acids, and H

Figure 3.2: Zinc Finger Structure

The Zinc finger C2H2 type domain structure from PDBsum [55].



```
C.{2,4}C.{3}[LIVMFYWC].{8}H.{3,5}H
```

Conserved patterns or motifs in proteins can be represented in different ways that capture different information about the pattern discovered. These patterns can be largely divided into those that are probabilistic and those that are deterministic. In addition, there are many patterns that are hybrids of the two approaches. They may use some probabilistic principles without fully invoking a probabilistic model (such as alignments and some early position-specific scoring matrices).

There is a large literature on both deterministic and probabilistic sequence patterns. The topics and techniques discussed here will be those that most pertain to the goal of high-throughput sequence analysis and do not represent an exhaustive covering of sequence pattern research.

The simple patterns (shown in italics and larger font in Figure 3.4) contained in the example sequences may be represented using various types of patterns.

3.1.1 Deterministic Patterns

Deterministic patterns [22] are defined by an exact pattern. A given sequence either matches or does not match a deterministic pattern. A common class of deterministic

Figure 3.3: Zinc Finger Sequence

The Zinc finger C2H2 type domain from PDBsum [55]. The helices annotated in the sequence correspond to the barrels seen in the structure from Figure 3.2.

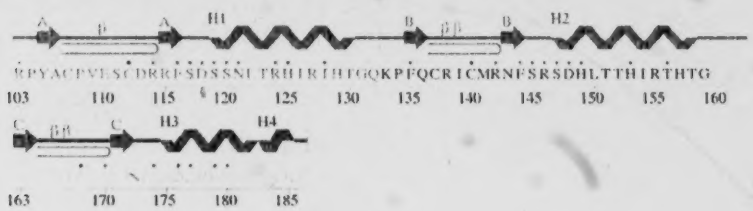


Figure 3.4: Patterns in Example Sequences

>sequence 1
QIKDLLVSSSTDLDTTLVLRENVATLPAEKMKPF*FIND*AF
*THE*KW*PATTERNS*FHVTILELKYFQESKPVMPQMMCNS
>sequence 2
RRVKVYLPQMKIEEKYNLTSVLMALGMTDLFIPSANLT*FI*
*ND*MF*THE*ED*PATTERNS*KISQAGSSAESLIGVIEDIKHSP
>sequence 3
ISEEYISYGGEKKILAIQGALEKALRWASGESFIELSNHK
FDRM*FIND*RK*THE*KL*PATTERNS*SAKFRRFT
>sequence 4
AKLAEQAERYDDNLLSVAYKNVVGARRSSWRVISSIEQKT
ERNEKKQQMGKEYREKIEAELQDICNDVLELLDK*FIND*MK
*THE*KL*PATTERNS*YLIPNRSQPESKVFYLKMKGDYFRYLS
EVASGDNKQTVSNSQQAYQEAFEISKKEMQPT
>sequence 5
MITILEKISAIESEMARTQKNKATSAHLGGGGTGEAGFEV
AKTGDARVGFVGFPSVGKSTLLSNLAGV*FIND*AN*THE*LR*P*
*ATTERNS*YSEVAAYEFTTLTTVPGCIKYKGAKIQLLDLPG
IIEGAKDGKGRGRQVIAVARTC

---

pattern is *regular patterns*, that can be expressed as *regular expressions*. Regular expressions allow

- exact matches (e.g. `FINDAFTHEKWPATTERNS`),
- wild-card matches (e.g. `FIND..THE..PATTERNS`),
- alternate matches (e.g. `FIND[AMR][FKN]THE[KEL][WDLR]PATTERNS`),
- variable-length matches (e.g. `FIND.{1,2}THE[KELR]{1,2}PATTERNS`) and
- arbitrary-length matches (e.g. `FIND[N]*THE.*PATTERNS`).

Other more powerful classes of deterministic patterns can be expressed by context-free grammars or Turing-equivalent languages. These classes of patterns have more power of expression than regular patterns but are also very computationally expensive to learn and recognize. Their biological significance may also be more difficult to interpret because of their complexity. The deterministic patterns discussed here will be restricted to a subset of regular patterns.

3.1.2 Probabilistic Patterns

Probabilistic sequence patterns [34] may be of basically the same structures as deterministic patterns except that they define the *probability* of a match rather than an explicit boolean match. The probability of a match between a protein sequence and the pattern is typically built up from the combination of the probabilities of a particular amino acid matching a particular part of the pattern. As with deterministic patterns, increased pattern power and complexity brings increased computational expense.

The main probabilistic pattern classes involve

- fixed-length matches,
- variable-length matches, and
- regular-expression equivalent matches.

Probabilistic patterns are often viewed as portions of sequences that are generated from a probabilistic model or distribution. We might view the majority of proteins as being generated from a 'background' distribution and the proteins with the function of interest being generated from a specific pattern distribution. Our goal in pattern recognition can be viewed as the task of determining with high confidence which distribution is most likely to have 'generated' the sequence. In the

Figure 3.5: Common Substrings

Common substrings from the example sequences.

```
FIND THE PATTERNS
```

---

ideal case, the statistical significance of a sequence belonging to a particular distribution would be *diagnostic* for the associated protein function (ie. the positive and negative groups would be completely separated into high and low probabilities when matched to the pattern model).

3.2 Deterministic Sequence Patterns

3.2.1 Common Substrings

Common substrings are the simplest form of sequence pattern (see Figure 3.5). The problem is conceptually very simple and relatively well-studied. As common subsequences are also very useful for a variety of applications outside of biological sequence analysis, many efficient solutions are available [40].

Many efficient algorithms exist for substring matching problems, including the Boyer-Moore [21], Knuth-Morris-Pratt [52], and Aho-Corasick [2] algorithms.

Suffix trees are of particular interest in exact string matching. Suffix trees (discussed in detail in Section 4.1 and very well developed by Gusfield [40]) are data structures that allow the acceleration of many string processing and pattern finding tasks (when applicable, these will be mentioned below). Of particular interest is the fact that suffix trees may be used to find the longest common subsequence of a set of sequences in time that is linear in the total number of symbols in the sequences. This algorithm and variations on it allow signature substrings to be found quickly in sequence families. Tree structures in general are often used as indices in order to accelerate a variety of lookup tasks on many algorithms such as database searches.

Although common substrings can be very useful patterns that are easily interpretable and efficient to work with, they have practical limitations. As biological sequences tend to mutate, there are very few patterns that are maintained as common substrings without noise. Very efficient wildcard and k-mismatch variations of suffix tree algorithms have been developed to deal with approximate matching

Figure 3.6: Consensus Sequences

A consensus sequence derived from the example sequences. In the 10th position, K is the consensus symbol as it occurs more than any other amino acid at that position (2/5 times) in the example sequences. The consensus sequence does not exactly match any of the example sequences.

```
Consensus   FINDAFTHEKLPATTERNS
Position    12345678901234567890
```

[40]. Suffix tree algorithms have also been applied to a wider set of parameterized string matching[4] problems [40, 65].

3.2.2 Consensus Sequences

Consensus sequences are another simple form of sequence pattern that are closely related to string matching. Consensus sequences are substrings where each position holds the most frequently occurring symbol at that position in the pattern. Consensus sequences encode neither the frequency of the consensus symbol nor the frequencies of any other symbols at a given position (see Figure 3.6). Despite their appealing simplicity and widespread use [58], these pattern representations have been argued to be misleading and error-prone [83].

One example of the use of consensus sequences is the 'TATAA box', a promoter region in bacterial DNA sequences. This pattern was recognized by David Pribnow in 1975 [78] and has been cited often since that time [58]. As discussed by Schneider [83], the frequencies at the 'TAA' positions of the consensus are 0.49, 0.58, and 0.54. By requiring exact matching at the 'T' position, at least 51% of the known sequences would be missed in error (false negatives). By using the full sequence 'TATAAT' for exact string matching, only 14 of 291 documented occurrences would be found. In practice, partial matches to the consensus may be allowed to reduce the problems of an overly exclusive consensus.

Given a set of example sequences, the consensus can be calculated very efficiently. Matches or partial matches to the consensus sequence can also be found very efficiently. The exact and approximate string matching algorithms mentioned above (Section 3.2.1) may be used. Slightly modified scoring algorithms may be

[4]Parameterized approximate string matching involves finding all occurrences of a pattern that are within a threshold distance (defined by some distance measure) of a string.

applied to partial matches. Suffix trees have also been used to find slightly more flexible consensus patterns with variable gaps and mismatches [67].

3.2.3 Alignments

Sequence alignments allow more flexible representation of sequence patterns. Alignments may be considered in terms of minimizing the 'edit distance' – the amount of change required to make one sequence identical to another. The allowed changes are mismatches, insertions, and deletions. Each alignment scheme assigns a cost for each change and optimizes the alignment so that the total edit cost is minimized.

Although sequence alignment algorithms incorporate both deterministic and probabilistic elements, they will be presented here because of the natural similarity to exact and approximate string matching algorithms.

Similarities between two sequences may be shown by pairwise alignments (see Figure 3.7). Similarities between three or more sequences are represented by multiple alignments (see Figure 3.8).

Pairwise Alignments

Needleman and Wunsch [74] created the first automated global alignment algorithm that optimizes the alignment over the entire length of the two sequences. Numerous variations and improvements have occurred since that time [40]. Smith and Waterman introduced local sequence alignments [86] which find more highly conserved subsequences. Improvements in performance with respect to time [39] and space [69, 46] have also been published. Scoring schemes have been refined to include affine gaps [39] and amino acid substitution matrices. Affine gaps assign non-linear penalties to gaps based on the idea that *inserting* a gap 'costs' more than *extending* that gap. Amino acid substitution matrices, such as those developed by Dayhoff [31] and Henikoff and Henikoff [43] assign different penalties for substituting amino acids that are similar than for substituting amino acids that are very different. These matrices, although not explicit probabilities, are based on probabilistic principles [34].

BLAST In addition to developments in optimal sequence alignment, work on approximate alignments has been very important for practical use. The FASTA [61] and BLAST [4] algorithms both advanced the practicality of searching sequence databases for local alignments because of greatly increased search speed over optimal alignments. Both work on the principle of heuristic exclusion of sequence regions that are unlikely to produce good local alignments. BLAST has been the dominant sequence alignment program almost since it was first released because of its superior speed and a well-developed statistical interpretation of the results.

Figure 3.7: Pairwise Alignment

A global pairwise alignment of subsequences of example sequences 1 and 2 is shown below. In this simple alignment, matching amino acid residues get a score of 3, mismatches a score of -3, and gaps a score of -2.

```
Subsequence 1 FTHEKWPATTERNSFHVT
Subsequence 2 ANLTFINDMFTHEEDPATTERNSK

---------FTHEKWPATTERNSFHVT
         ||||  ||||||||
ANLTFINDMFTHEEDPATTERNS---K

Alignment Length = 27
Alignment Score = 3
```

Modifications to the algorithm, such as PSI-BLAST [5], have also been significant for finding more distantly related sequences.

BLAST returns the best alignments it can make between the query sequences and sequences in a database. The best alignments have high bit-scores and low Expectation Values (E-values). The lower the E-value, the less likely it is that an alignment as good as the one seen would occur by random chance.

Suffix trees have been used to accelerate pairwise alignments, especially in the case of k-difference inexact matching [40].

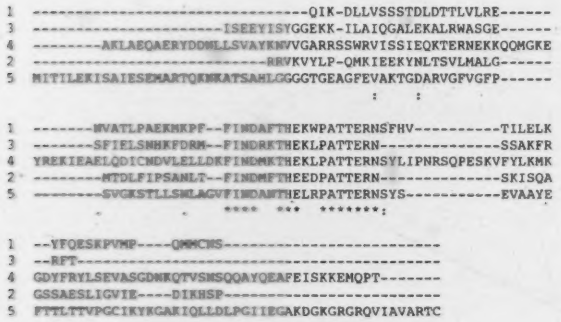
Genomic Alignments As genomic data has become available, whole genome alignments have become of interest. Slightly different alignment methods are used for these very large-scale problems. MUMmer [54], for example, uses suffix trees to allow the very efficient alignment of genomic DNA.

Multiple Alignments

Multiple sequence alignment is useful for displaying the commonalities in a family of sequences that may have common structure or function (see Figure 3.8). Although pairwise alignment may provide some of this information, it does not optimize the comparison across all the sequences in the family. Arthur Lesk (as quoted

Figure 3.8: Multiple Alignment

This is a multiple alignment of the example sequences as performed by the ClustalW program. '-' symbols represent gaps that have been inserted as part of the alignment.



by Gusfield [40]) said, 'One or two homologous sequences whisper ... a full multiple alignment shouts out loud'.

Optimal multiple sequence alignment is extremely costly (NP-hard for scoring schemes used by biologists [96]) and most practical methods use a variety of heuristics to perform multiple alignments. As with pairwise alignments, many variations and improvements have been made to multiple alignment methods [40]. Many other patterns, such as consensus sequences or HMMs [34], utilize multiple alignments as a preprocessing step.

ClustalW ClustalW [92] is a popular multiple sequence alignment program that illustrates the type of approximation algorithms that may be used to speed up the alignment. ClustalW first performs pairwise alignments between all the pairs of sequences to be aligned. The similarity scores are used to create a tree by clustering more similar sequences together. The sequences are then progressively aligned starting with the most similar sequences. Sequences are merged into profiles that can then be merged with other sequences or profiles. A variety of other heuristic refinements improve the performance of the algorithm.

By observing the aligned portion of the sequences, many patterns can be easily observed. The disadvantage of these patterns is that although a multiple alignment may assist in viewing a pattern in the sequences, it may still be difficult to define the pattern so that it may be found in other sequences. Some progressive alignment

methods allow the incorporation of a new sequence into a previous alignment.

Other Multiple Alignment Methods Gibbs sampling techniques have been used to accelerate multiple alignment [56]. This method is commonly used to find short local alignments that are motifs (see Section 3.3.1) in relatively divergent groups of sequences. Hidden Markov models may also be used for multiple sequence alignment [34].

3.2.4 Regular Languages

Although alignments are extensively used and intuitively satisfying, the importance of other types of patterns has been underscored. R.F. Doolittle stated (as quoted in the PROSITE user manual [84]),

> There are many short sequences that are often (but not always) diagnostics of certain binding properties or active sites. These can be set into a small sub-collection and searched against your sequence.

A.M. Lesk (also quoted in the PROSITE user manual [84]) mentioned,

> In some cases, the structure and function of an unknown protein which is too distantly related to any protein of known structure to detect its affinity by overall sequence alignment may be identified by its possession of a particular cluster of residues types classified as a motifs. The motifs, or templates, or fingerprints, arise because of particular requirements of binding sites that impose very tight constraint on the evolution of portions of a protein sequence.

For these reasons, other pattern types are also important for sequence analysis. Regular languages handle variation in biological sequences through the use of wild cards, equivalent sets of residues, and variable length gaps.

PROSITE The PROSITE [84] database and pattern matching tools find both deterministic patterns and probabilistic profiles. Deterministic PROSITE patterns take the form of regular-expression-like strings. They vary slightly from standard regular expression in notation, but retain most of the power. PROSITE patterns include the subset of regular expressions that allow

- exact matches
  (e.g. `F-I-N-D`),
- alternate matches
  (e.g. `F-I-N-D-[TG]-[KM]-T-H-E`),

Figure 3.9: PROSITE Output

Some PROSITE motifs found in the example sequences are shown below. Motifs occurring more than once have been removed for brevity.

```
>Seq 1 : PS00005 PKC_PHOSPHO_SITE Protein kinase C phosphorylation site.
     49 - 51  TeR
>Seq 1 : PS00006 CK2_PHOSPHO_SITE Casein kinase II phosphorylation site.
      9 - 12  SstD
     11 - 14  TdlD
     57 - 60  TilE
>Seq 2 : PS00001 ASN_GLYCOSYLATION N-glycosylation site.
     17 - 20  NLTS
     36 - 39  NLTF
...
>Seq 3 : PS00004 CAMP_PHOSPHO_SITE cAMP- and cGMP-dependent protein kinase phosphorylation site.
     68 - 71  RRfT
...
>Seq 4 : PS00003 SULFATION Tyrosine sulfation site.
      3 - 17  laeqaerYddnllsv
...
>Seq 4 : PS00007 TYR_PHOSPHO_SITE Tyrosine kinase phosphorylation site.
    110 - 118  KmkgDyfrY
...
>Seq 4 : PS00008 MYRISTYL N-myristoylation site.
     24 - 29  GArrSS
...
>Seq 5 : PS00017 ATP_GTP_A ATP/GTP-binding site motif A (P-loop).
     52 - 59  GfpsvGKS
>Seq 5 : PS00905 GTP1_OBG GTP1/OBG family signature.
    117 - 130  DLPGIIEGAkdGkG
```

- wild-card matches
  (e.g. `F-I-N-D-x(2)-T-H-E-x(2)-P-A-T-T-E-R-N-S`), and
- fixed-range variable-length matches
  (e.g. `T-H-E-x(2,5)-P-A-T-T-E-R-N-S`).

PROSITE patterns are created manually through the observation of functional regions (typically well-known) in multiple alignments of protein sequences. As a result, the database is fairly small in size (about 1300 pattern entries) but still useful for identifying a number of common protein patterns. A simple tool such as ScanProsite [38] searches a given set of protein sequences for PROSITE patterns by converting the pattern to a regular expression and using programs such as Perl [95] to perform the search. Figure 3.9 shows some of the results obtained from PROSITE on our example sequences. The numbers of the form `PS#####` are PROSITE accession numbers.

TEIRESIAS TEIRESIAS [80] is a combinatorial pattern discovery algorithm developed and patented by researchers at IBM. It discovers fixed-length patterns with

Figure 3.10: TEIRESIAS Output

TEIRESIAS run on the example sequences using the options shown.

Using the default options: Pattern Length=10, Template Length=12, Minimum Support=5 and Convolution Length=5.

```
THE..PATTERN
HE..PATTERNS
```

Using the options: Pattern Length=3, Template Length=3, Minimum Support=5 and Convolution Length=2.

```
PATTERNS
TERN
FIND
THE
```

Using the options: Pattern Length=3, Template Length=5, Minimum Support=5 and Convolution Length=2.

```
FIND..THE..PATTERNS
TERN
EK..A
T..NS
```

---

wild-cards. Although used for a variety of sequence types, it has been used with biological sequences to efficiently find and report patterns.

The concept underlying the TEIRESIAS algorithm is the *a priori* principle from association-rule data mining techniques [41]. The algorithm first finds frequently occurring smaller 'elementary' patterns of a user specified length (the 'scanning' phase). Then, longer patterns are built up by combining (or 'convoluting') compatible shorter patterns into progressively longer ones until the support (number of occurrences) for the new longer patterns would be less than for the shorter ones (the 'convolution' phase). In this way the algorithm ensures the greatest pattern specificity while including all possible occurrences within the training set.

The algorithm outputs the patterns and the positions of the occurrences of those patterns in the training sequences. These patterns are fixed-length patterns that allow only specific amino acid residues or wild-card characters at each position of the pattern. As such, they are quite inflexible and small amounts of 'noise' or variation in a pattern will often generate a number of very similar patterns. Users must specify the number of non-wild-card characters in the pattern, the maximum extent of an elementary (ungapped) pattern, the minimum allowed support for a pattern, and the number of overlapping symbols in the convolved pattern.

The TEIRESIAS output can be very sensitive to the search parameters. Using

Figure 3.11: PRATT Output

```
Best Patterns (after refinement phase):
          fitness     hits(seqs)    Pattern
A   1:    62.5508       5(   5)   F-I-N-D-x(2)-T-H-E-x(2)-P-A-T-T-E-R-N-S
B   2:    58.3807       5(   5)   I-N-D-x(2)-T-H-E-x(2)-P-A-T-T-E-R-N-S
C   3:    54.2107       5(   5)   N-D-x(2)-T-H-E-x(2)-P-A-T-T-E-R-N-S
```

the defaults, for example, it does not achieve optimal results on our example data set. Figure 3.10 shows various results obtained on our example sequences by changing the algorithm parameters. With some parameter tuning, the algorithm recovers the expected patterns. In general, however, we do not know the patterns we would like to discover. This makes parameter choice more difficult in practice.

Although TEIRESIAS runs quickly compared to many previous algorithms of this type, the running time increases proportionally with the size of the output. For less stringent parameter settings, the output size can grow very quickly (more than linear in size of the training data). In addition, proper parameter selection is crucial for good results. For these reasons, TEIRESIAS is not well suited to automated prediction.

PRATT PRATT [49, 48] is a graph-based pattern discovery algorithm. It supports the discovery of regular-expression-like patterns that are capable of expressing exact matches, alternate matches, and wild-card matches of variable length. The two stage process involves an initial discovery phase and a second evaluation and pattern refinement phase. In the discovery phase, very simple fixed-length patterns with occurrences in the training sequences are found. This is done by extending very short patterns discovered with wildcard and amino acid symbols and counting the number of occurrences. A graph structure is used to accelerate the search. Patterns that are given a high score by the algorithm are then passed to the refinement stage in which they are combined in cases where the resulting combination (through the addition of ambiguous symbols) has a higher fitness score.

The output is a set of patterns that are equivalent to PROSITE patterns and may be used in the same way.

Figure 3.11 shows the results of PRATT on the example sequences. With the default parameters PRATT finds the pattern `FIND..THE..PATTERNS`.

Pratt discovers many patterns that, despite the refinement phase, contain many

redundant results (as seen in Figure 3.11). PRATT is very computationally expensive and also sensitive to the input parameters (though not as inflexible as TEIRESIAS).

SPLASH The patterns discovered by the SPLASH [25] algorithm include those found by PRATT. SPLASH additionally allows more flexibility by using symbol similarity metrics rather than exact matches. In the case where identity (exact matching) is used, SPLASH has been shown to be much faster and more scalable (sub-linear time complexity in the size of the database) than PRATT (super-linear time complexity with the size of the database). Where similarity metrics are allowed, the discovered patterns have been shown to be more sensitive than those obtained with exact matching.

SPLASH and TEIRESIAS are both proprietary IBM algorithms.

3.3 Probabilistic Sequence Patterns

3.3.1 Probabilistic Profiles

Position-specific scoring matrices (PSSMs) are common probabilistic models of sequence patterns. These profiles are fixed-length patterns represented by a scoring matrix (see Figure 3.12). The columns in the matrix correspond to sequence positions and rows correspond to particular symbols. The score at row i and column j of the matrix represents the score given if amino acid i occurs at position j of the pattern. The pattern score is evaluated at each position of the sequence by moving the pattern over the sequence and reevaluating the score at each position. Each position in the sequence is assigned a score by this moving window. Scores above a certain threshold are considered to be matches. The scores are typically derived from probabilities and may be smoothed or scaled for practical reasons. Log ratios are often used to make the calculation more efficient and to avoid floating point underflow errors. Figure 3.12 shows typical example calculations for creating the PSSM (used in the MEME/MAST system [10, 12, 11]). PSSMs for other applications may be calculated in slightly different ways.

Figure 3.13 illustrates some example calculations of scores along the example sequences. The scores obtained from the PSSM at each point in the sequence can also be converted to p-values which indicate the probability that a random sequence (from the background probabilities) would have an equal or greater score.

An approximation of the PSSM from Figure 3.12 can be illustrated by the following multi-level consensus (with second-most common residues also shown).

```
FINDAF
  A MN
```

Figure 3.12: Position-Specific Scoring Matrix (PSSM)

The position-specific probability matrix is calculated from counts of the amino acids seen at motif occurrences in the training sequences (see Figure B.1). These counts are used to calculate a position-specific probability matrix (top). The position-specific scoring matrix (bottom) is calculated from the position-specific probability matrix by taking the base 2 log of the ratio of the probability of the symbol in the motif (from the PSPM) and the probability of the symbol in the background (not shown). Each entry is then multiplied by 100 and rounded to the nearest integer. This output is the result of using MEME [10] on the training sequences (see Figure B.1) with patterns similar to those found in the example sequences.

Position-Specific Probability Matrix

	1	2	3	4	5	6
A	0.166	0	0.333	0	0.5	0
D	0	0	0	1.000	0	0
F	0.833	0	0	0	0.166	0.5
I	0	0.833	0	0	0	0
M	0	0	0	0	0.333	0.166
N	0	0.166	0.666	0	0	0.333

Position-Specific Scoring Matrix

	1	2	3	4	5	6
A	**-151**	-291	**142**	-347	**230**	-243
C	-123	-171	-124	-266	-25	-111
D	-381	-320	-98	**449**	-309	-274
E	-468	-415	-279	-149	-344	-346
F	**431**	-237	-380	-417	**134**	**351**
G	-436	-416	-210	-373	-251	-353
H	-292	-296	-102	-194	-175	-113
I	-234	**345**	-371	-420	-77	-138
K	-470	-371	-282	-422	-321	-314
L	-129	-28	-371	-414	-44	-86
M	-205	-83	-344	-399	**281**	**181**
N	-405	**-32**	**368**	-89	-280	**196**
P	-335	-367	-263	-395	-276	-298
Q	-410	-330	-211	-321	-236	-240
R	-422	-347	-269	-367	-263	-271
S	-318	-327	-135	-330	-162	-260
T	-388	-261	-208	-386	-172	-241
V	-214	119	-268	-352	-25	-106
W	-59	-220	-262	-277	-102	-14
Y	-38	-266	-322	-359	-154	26

Figure 3.13: Using a Position-Specific Scoring Matrix

The PSSM from Figure 3.12 is used to scan the sequences and calculate the scores at each position. Given a score threshold of 0, the top calculation results in a match to the PSSM at the position shown. The bottom calculation does not result in a match to the PSSM in Figure 3.12.

```
          FINDMF
          ||||||
GMTDLFIPSANLTFINDMFTHEEDPATTERN
```

$$\begin{aligned} PSSM(FINDMF) &= PSSM(F,1) + PSSM(I,2) + PSSM(N,3) \\ &\quad + PSSM(D,4) + PSSM(M,5) + PSSM(F,6) \\ &= 431 + 345 + 368 + 449 + 281 + 351 \\ &= 2225 \end{aligned}$$

```
       GIADRK
       ||||||
NHKFDRMGIADRKTEKLGMTDLFIFINDMFT
```

$$\begin{aligned} PSSM(GIADRK) &= PSSM(G,1) + PSSM(I,2) + PSSM(A,3) \\ &\quad + PSSM(D,4) + PSSM(R,5) + PSSM(K,6) \\ &= -436 + 345 + 142 + 449 + -263 + -314 \\ &= -77 \end{aligned}$$

Suffix trees (see Section 4.1) have been used to accelerate predictions using PSSMs [33].

PROSITE Profiles PROSITE profiles [84] are simple PSSMs that have been created from multiple sequence alignments. They are considerably more computationally expensive to recognize than PROSITE patterns. Each profile is developed through extensive human involvement.

BLOCKS The BLOCKS [44] database contains a series of multiple alignments representing highly conserved representatives of sequences matching the PROSITE database. The regions are selected automatically and then calibrated against the SwissProt database [19] to determine the probability of obtaining a random match.

MEME The MEME [10] tool discovers a PSSM from a set of unaligned sequences. MEME was used to find the example PSSM in Figures 3.12 and 3.13.

MEME applies the *expectation maximization* (EM) algorithm (which is described in machine learning texts [42, 70]). The EM algorithm is used to separate instances from two unknown probabilistic distributions. In this case, two distributions are the 'motif' distribution (which 'generates' occurrences of the motif) and the 'background' distribution (which 'generates' all other amino acid residues in the training set). Given a user-specified motif width w, MEME creates a list of all the substrings of width w in the training sequences. Each substring must be assigned to either the 'motif' model or the 'background' model. The expectation maximization is used to iteratively update in two phases, *E* and *M*.

- the *E*-step: Calculates the assignments ('motif' or 'background') of the substrings given the current model parameters and
- the *M*-step: Updates the model parameters given the substring assignments.

Through repeated iterations of the two phases, the algorithm converges on a motif model and substrings that match it. Motifs of various lengths can be found by rerunning the algorithm for each possible motif width. Multiple motifs can be found by masking out regions that match motifs that have already been found and rerunning the algorithm. In this way, if is possible (although computationally expensive) for MEME to find a specified number of patterns in a specified range of widths.

PSI-BLAST PSI-BLAST [5] augments the search for homologous sequences by using a PSSM that is automatically created from a multiple alignment of high-scoring BLAST alignments. The PSSM is then iteratively realigned to the sequence database and the values are refined by the high scoring alignments from each iteration. This Position-Specific Iterative (PSI) approach to BLAST allows increased

sensitivity in finding related sequences as it assigns more weight to highly conserved regions and less weight to regions with significant sequence divergence.

Gibbs Sampling The Gibbs sampler [56, 93] finds motifs by using a statistical sampling technique, Gibbs sampling, to efficiently find fixed-length motifs. Gibbs sampling can be seen as a stochastic variation of the expectation maximization (EM) algorithm (above).

3.3.2 Markov Models

Markov Chains

Classical Markov chains have been used for many sequence analysis applications. Markov chains are used to define the probability of the next symbol in a sequence given the preceding symbols (or history). The model is simplified by assuming that the distribution has a 'short memory' – that is, it is not necessary to keep statistics for histories beyond a certain length. This is the 'Markov assumption'. Markov first proposed the technique to predict whether the next letter in Pushkin's *Eugene Onegin* would be a vowel or a consonant [66].

Markov chains have been used for a variety of applications. For example, N-gram techniques which have long been used in natural language processing are simple Markov chains [50]. N-gram models make use of the probability of a word occurring in a text given the N previous words. Markov chains are used in biological sequence analysis in a manner similar to Markov's technique - we make use of the probability of a certain symbol given the previous symbols [34]. The *order* of the model is equal to the number of preceding symbols used.

For use in classification of sequences, we can use the following first-order Markov chain to define the probability of the sequence $x = x_1 \dots x_m$.

$$
\begin{aligned}
P(x) &= P(x_1 x_2 \dots x_{m-1} x_m) && (3.1)\\
&= P(x_1)P(x_2|x_1)P(x_3|x_1x_2)\dots \\
&\quad P(x_{m-1}|x_1 \dots x_{m-2})P(x_m|x_1 \dots x_{m-1}) && (3.2)\\
&= P(x_1)P(x_2|x_1)P(x_3|x_2)\dots \\
&\quad P(x_{m-1}|x_{m-2})P(x_m|x_{m-1}) && (3.3)\\
&= P(x_1)\prod_{i=2}^{m} P(x_i|x_{i-1}) && (3.4)
\end{aligned}
$$

The simplification step from Equation 3.2 to Equation 3.3 is possible because of a first-order Markov assumption. We estimate the values of $P(x_i|x_1 \dots x_{i-1})$ by assuming that only the previous symbol x_{i-1} is important to the distribution of

the current symbol x_i. Although this may seem an outrageous assumption in a biological context, it allows us a simple and efficient model for classification that can be useful in some cases.

To use a first-order Markov chain for classification of a given protein function y, we calculate the probability of a sequence x given the positive and negative training sequences. The sequences that are known to have the function being modeled are used to obtain the positive $(+)$ model parameters and the sequences that are known to not have the function are used to obtain the negative $(-)$ model parameters. These parameters are estimated by the count $C(a)$ of the number of occurrences of the substring a in the training data.

$$P(x_i|x_{i-1}) = \frac{P(x_{i-1}x_i)}{P(x_{i-1})} \tag{3.5}$$

$$= \frac{C(x_{i-1}x_i)}{C(x_{i-1})} \tag{3.6}$$

An example set of model parameters is found in Table 3.1. The parameters were generated from the example training sequences in Figure B.2 (found in Appendix B). In this example we convert the example protein sequences to a smaller alphabet[5]. The mapped training sequences are found in Figure B.3. Mapping of the amino acid alphabet to simpler alphabets based on physical similarities has been fruitful in a variety of applications [72, 6] (and reduces the space required for this example).

Given the model parameters and a test sequence x, we may calculate the likelihood of the sequence given the positive model $P_+(x)$ and the negative model $P_-(x)$. These values are often combined into a single score as a *log-odds ratio*.

$$\textit{log-odds ratio} = \log\left(\frac{P_+(x)}{P_-(x)}\right) \tag{3.7}$$

For convenience, we can build a single discriminative model by combining the parameters of the positive and negative models into log-odds ratios (as in the bottom of Table 3.1). A log-odds ratio can be determined for each position x_i of the sequence x and the sum of these ratios is equal to the total log-odds ratio for the sequence.

$$\textit{position log-odds ratio} = \log\left(\frac{P_+(x_i|x_{i-1})}{P_-(x_i|x_{i-1})}\right) \tag{3.8}$$

[5]In this particular alphabet [72], L represent the hydrophobic amino acids (LVIMC), F represents the hydrophobic/aromatic side chains (FYW), A represents the polar amino acids (AGSTP), and E represents the hydrophilic amino acids (EDNQKRH). Example mappings can be seen in Table B.1.

Table 3.1: Markov Chain Probabilities

The probabilities found in the following tables were calculated from the training sequences in Figure B.2. The parameters for the top, middle, and bottom matrices are for the positive, negative, and combined discriminative models respectively.

Positive Training Data

	$P_+(\cdot\|A)$	$P_+(\cdot\|E)$	$P_+(\cdot\|F)$	$P_+(\cdot\|L)$
A	18/37	13/50	4/11	5/18
E	12/37	28/50	0/11	8/18
F	6/37	2/50	1/11	3/18
L	1/37	7/50	6/11	2/18

Negative Training Data

	$P_-(\cdot\|A)$	$P_-(\cdot\|E)$	$P_-(\cdot\|F)$	$P_-(\cdot\|L)$
A	44/125	36/144	5/29	42/99
E	46/125	53/144	14/29	29/99
F	10/125	13/144	3/29	3/99
L	25/125	42/144	7/29	25/99

Log-odds Ratios

	$\log \frac{P_+(\cdot\|A)}{P_-(\cdot\|A)}$	$\log \frac{P_+(\cdot\|E)}{P_-(\cdot\|E)}$	$\log \frac{P_+(\cdot\|F)}{P_-(\cdot\|F)}$	$\log \frac{P_+(\cdot\|L)}{P_-(\cdot\|L)}$
A	0.3236	0.0392	0.7463	-0.4235
E	-0.1263	0.4197	$-\infty$	0.4169
F	0.7066	-0.8140	-0.1292	1.7047
L	-2.0015	-0.7340	0.8152	-0.8210

$$\textit{log-odds ratio} = \log\left(\frac{P_+(x)}{P_-(x)}\right) \tag{3.9}$$

$$= \sum_i \log\left(\frac{P_+(x_i|x_{i-1})}{P_-(x_i|x_{i-1})}\right) \tag{3.10}$$

The log-odds ratio gives a score at each position that allows us to better observe the contribution of each amino acid in the sequence to the overall classification.

We can predict whether a protein is in the positive or negative class by using the calculated probabilities. Consider the test sequence 'KPF*FIND*AF', which is mapped to 'EAF*FLEE*AF'. To calculate the log odds of this sequences, we calculate

$$\begin{aligned}
lo(x_i|x_{i-1}) &= \log\left(\frac{P_+(x_i|x_{i-1})}{P_-(x_i|x_{i-1})}\right) \\
\log\left(\frac{P_+(x)}{P_-(x)}\right) &= \sum_i lo(x_i|x_{i-1}) \\
&= lo(A|E) + lo(F|A) + lo(F|F) + lo(L|F) \\
&\quad + lo(E|L) + lo(E|E) + lo(A|E) + lo(F|A) \\
&= 0.0392 + 0.7066 + -0.1292 + 0.8152 \\
&\quad + 0.4169 + 0.4197 + 0.0392 + 0.7066) \\
&= 3.0142
\end{aligned}$$

Because the log-odds score is greater than 0, this sequence is more likely to have been generated by the positive model and will be predicted to be positive. Now consider another test sequence 'ISEEYI' which is mapped to 'LAEEFL'.

$$\begin{aligned}
\log\left(\frac{P_+(x)}{P_-(x)}\right) &= lo(A|L) + lo(E|A) + lo(E|E) \\
&\quad + lo(F|E) + lo(F|L) \\
&= -0.4235 + -0.1263 + 0.4197 \\
&\quad + - 0.8140 + 0.8152 \\
&= -0.1289
\end{aligned}$$

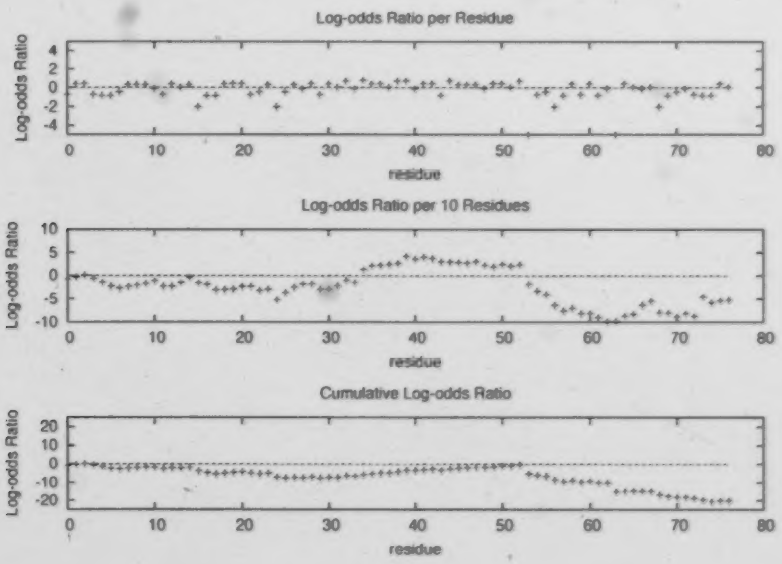
As the log-odds score for this sequence is less than 0, it is more likely to have been generated by the negative model and will be classified as such.

These examples illustrate a *global* prediction (across the entire sequence). We may also have reason to believe that only part of our sequence matches the positive model. For this problem we would like a *local* prediction.

We calculate the results for the example sequences from Figure 3.1. Figure 3.14 displays the log-odds ratios for example sequence 1. The top plot shows the log-odds ratio at each residue. From that plot we get little clear indication of which

Figure 3.14: Example Markov Chain Predictions

A positive log-odds score for a particular position or region means that it is more likely to have the function than not.



model the sequence matches better. The bottom plot shows the cumulative log-odds ratio over the entire sequence. It appears from the cumulative (global) score that this sequence matches the negative model more closely than the positive model. The middle plot, however, displays the sum of log-odds ratios at each position over the previous 10 amino acids. This visualization helps us select a region of the sequence which does appear to match the positive model. The residues that have a positive score over a 10-window are the residues 34-52 in example sequence 1. This region corresponds to the string 'FINDAFTHEKWPATTERNS' in the original sequence. As illustrated by this example, Markov chains can be used for both global (entire sequence) and local (partial sequence) discrimination. The results provide some intuition about which regions are most likely to be responsible for the protein function of interest.

Many extensions have been made to first-order Markov chains shown here. Higher order Markov chains have been used for a variety of purposes. Combinations of Markov models of various orders (Inhomogeneous, Interpolated Markov Models or IMMs) have been used very effectively in the GLIMMER [32] program for prokaryotic gene prediction. The very similar Variable Memory Markov (VMM) models [16] have been used to select a variable history length (or model order) depending on the residues preceding the current amino acid (the 'context' of the current amino acid). Probabilistic suffix trees [16] have been used to both reduce memory requirements and accelerate the speed of VMMs. Both VMMs and PSTs are discussed in more detail in Section 4.2.

Because of the simple interpretable model and speed with which predictions can be made, Markov chains have potential as models for high-throughput protein function prediction. Unfortunately, they have not been developed for 'pattern discovery' and so require that training data be labeled globally. In addition, the local prediction demonstrated above requires prediction scoring using a sliding window of a fixed size. The optimal window size may vary from one functional pattern to another. Despite these limitations, Variable Memory Markov models are attractive for the protein function prediction task, especially as their time requirements can be minimized when they are implemented as PSTs.

Hidden Markov Models

Hidden Markov models (HMMs) are powerful and popular predictive models. HMMs are equivalent to probabilistic regular expressions and allow more sensitivity than exact regular expression matching. They also add more flexibility than fixed length position-specific scoring matrices (PSSMs). While we will try to provide some intuition for the use of HMMs in the following section, interested readers should consult a more complete treatment of the topic (such as Durbin *et al.* [34]).

While HMMs make use of the same Markov assumption as simpler Markov

chains, they also incorporate the notion of state. For an example of state, we may consider each position *observed* in a protein sequence x_i to be associated with a *state* π_i. Here, the state can take either of two values $\pi_i \in \{+, -\}$, a functionally important region $(+)$ or a functionally unimportant region $(-)$. As we see in the example sequence below, each position corresponds to a state and an amino acid symbol.

Sequence	x	AELQDICNDVLELLDKFINDMKTHEKLPATTERNSYLIP
Hidden States	π	----------++++++++++++++++++++-----

The Markov assumption applies because the probability of reaching the current state π_i is dependent only upon the previous state π_{i-1}. This conditional probability is called the *transition* probability.

$$P_{transition} = P(\pi_i|\pi_{i-1}) \tag{3.11}$$

The amino acid symbols at each position x_i are then emitted with a certain probability that is dependent only on the state π_i at that position. These conditional probabilities are called *emission* probabilities.

$$P_{emission} = P(x_i|\pi_i) \tag{3.12}$$

These states, transition probabilities, and emission probabilities can be combined into a single model called a hidden Markov model (HMM). A simple model for our example situation is shown in Figure 3.15. The model structure – consisting of the states, symbols, and their associated probabilities – are very specific to each application (Figures 3.16 and 3.17 show two other more complex model structures).

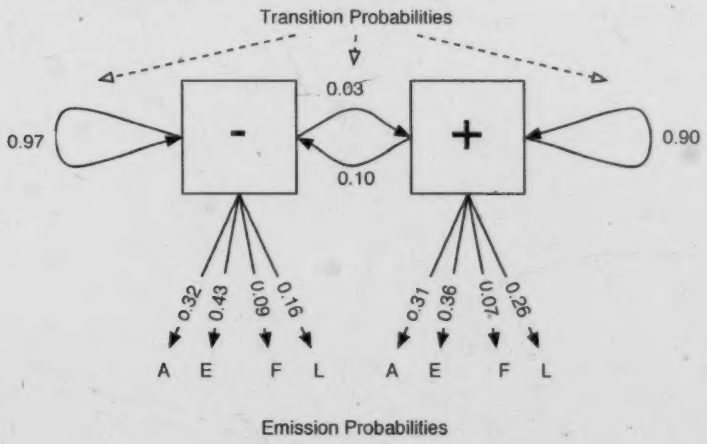
When given a new protein sequence, we can see the amino acid x_i at each position i but we cannot observe the corresponding state π_i. It is these states that are 'hidden' and that we seek to predict by using the HMM. In our example, we would like to predict which regions are functionally important ('+' states) and which are not ('-' states)[6].

The parameters of the model for the transition and emission probabilities can be learned in several ways. They are most easily learned when we are provided with a set of training sequences for which the symbols and corresponding states are completely specified. In the example, we might obtain training sequences that have

[6]Some readers may correctly object that our simple HMM in Figure 3.15 is very unlikely to repeatedly produce ordered symbols of the form 'FIND..THE..PATTERNS'. Despite this, we will use this pattern as an instructional example for continuity. Profile HMMs, which are discussed later, do take advantage of positional information within the motif and are able to better model this type of motif.

Figure 3.15: An Example HMM

The positive and negative sequence positions each correspond to a state in the model (+ or -). The amino acids are emitted with different probabilities for each state.



been labeled according to the results of biological testing. Although this simplifies the learning process, training data is not always available in this way. The Baum-Welch algorithm, an application of expectation maximization (EM), can be used to train on unlabeled training data. A variation on this, called Viterbi training, may also be used. For biological sequences, multiple alignments of relevant sequence regions can provide the needed information for parameter estimation. These techniques are discussed elsewhere. Much of the current discussion is based on Durbin *et al.* [34].

Once we have obtained the transition and emission probabilities that define the model, we can define the joint probability for an entire sequence of symbols and states.

$$P(x,\pi) = P(\pi_1|\pi_0)\prod_{i=1}^{L} P(x_i|\pi_i)P(\pi_i|\pi_{i-1}) \tag{3.13}$$

With this knowledge, the HMM may be used to calculate information about the sequence which is crucial to the protein function prediction task.

The Most Probable State Path The most probable state path $\pi*$ is the single most probable path of states π through the model given the sequence x. The calcu-

lation of the most probable state path arises from our definition of the joint probability of symbols and states for the sequence in Equation 3.13.

$$\pi* \quad = \quad argmax_{\pi} P(x, \pi) \tag{3.14}$$

Although the number of possible paths increases exponentially with the length of the sequence, we can efficiently calculate the most probable state path $\pi*$ using dynamic programming. By defining the probability of the most probable state path ending at a certain position i with a specific state k we can calculate the probability of the most probable path ending at the next position $i+1$ in a certain state l.

$$v_l(i+1) \quad = \quad P(x_{i+1}|\pi_{i+1} = l) max_k \{v_k(i) P(\pi_{i+1} = l|\pi_i = k)\} \tag{3.15}$$

The Viterbi algorithm uses this recursive equation to calculate the most probable path ending at each point in the sequence with a current state k. By taking the most probable path ending at the last position of the sequence we can obtain the most probable path of states through the entire sequence without evaluating every possible path individually. An example of this calculation (without numbers) for the sequence fragment $\ldots x_{16}x_{17}\ldots = \ldots KF\ldots$ follows.

i	$\ldots$	16	17	$\ldots$
x	$\ldots$	$x_{16} = K$	$x_{17} = F$	$\ldots$
$v_+(i)$	$\ldots$	$v_+(16)$ $\begin{matrix}\rightarrow\\ \searrow\end{matrix}$	$v_+(17) = P(F\|+) max \left\{ \begin{matrix} v_+(16)P(+\|+) \\ v_-(16)P(+\|-) \end{matrix} \right\}$ $\begin{matrix}\rightarrow\\ \searrow\end{matrix}$	$\ldots$
$v_-(i)$	$\ldots$	$v_-(16)$ $\begin{matrix}\nearrow\\ \rightarrow\end{matrix}$	$v_-(17) = P(F\|-) max \left\{ \begin{matrix} v_+(16)P(-\|+) \\ v_-(16)P(-\|-) \end{matrix} \right\}$ $\begin{matrix}\nearrow\\ \rightarrow\end{matrix}$	$\ldots$

After calculating the Viterbi values $v_k(i)$ in the forward direction, the algorithm traces back over the sequence through pointers that indicate from which state the maximum value was obtained at each step. The path back through the states starting at the maximum $v_k(i)$ at the end of the sequence is the Most Probable State Path (MPSP) path. In the example below, D is the final amino acid and the maximum v_k is v_+. The traceback through states is indicated by parentheses and the most probable state path is indicated below.

i	$\ldots$	16	17	18	19	20
x	$\ldots$	K	F	I	N	D
$v_+(i)$	$\ldots$	v_+	v_+	$(v_+) \rightarrow$	$(v_+) \rightarrow$	(v_+)
$v_-(i)$	$\ldots$	$(v_-) \rightarrow$	$(v_-) \nearrow$	v_-	v_-	v_-
$MPSP$	$\ldots$	$-$	$-$	$+$	$+$	$+$

Probability of a Sequence For protein function prediction we often want to know the probability that each sequence in a test set belongs to a family of proteins. If we have an HMM representing that sequence family, we can calculate the probability of each sequence given the model.

$$P(x) = \sum_{\pi} P(x, \pi) \tag{3.16}$$

Although the number of possible state paths π through the model increases exponentially with the length of the sequence, we can again use dynamic programming to calculate this probability efficiently. The *forward algorithm* defines the probability of being in a specific state k at a certain position i given the symbols up to and including this position.

$$f_k(i) = P(x_1, \ldots, x_i, \pi_i = k) \tag{3.17}$$

In order to calculate the probability of the entire sequence, we can recursively calculate $f_k(i)$ using the following equation which incorporates the emission and transition probabilities.

$$f_l(i+1) = P(x_{i+1}|\pi_{i+1} = l) \sum_k f_k(i) P(\pi_{i+1} = l|\pi_i = k) \tag{3.18}$$

The forward values $f_k(i)$ are calculated during a forward pass through the sequence in a manner similar to the Viterbi algorithm, except that the sum over the previous states is taken rather than the maximum.

i	...	16	17	...
x	...	$x_{16} = K$	$x_{17} = F$	...
$f_+(i)$	...	$f_+(16)$ $\rightarrow$ $\searrow$	$f_+(17) = P(F\|+)$ ($f_+(16)P(+\|+)$ $\rightarrow$; $+ f_-(16)P(+\|-)$ $\searrow$)	...
$f_-(i)$	...	$f_-(16)$ $\nearrow$ $\rightarrow$	$f_-(17) = P(F\|-)$ ($f_+(16)P(-\|+)$ $\nearrow$; $+ f_-(16)P(-\|-)$ $\rightarrow$)	...

Using this equation, the probability of the sequence x of a certain length L can be calculated. It is the final result of the forward algorithm at the *end* of the sequence.

$$P(x) = \sum_k f_k(L) P(end|x_k) \tag{3.19}$$

The probability can be calculated in this forward direction ($1 \ldots L$) or in the backward direction ($L \ldots 1$) using the analogous (but not exactly equal) *backward algorithm* and its definition.

$$b_l(i) = P(x_{i+1}, \ldots, x_L | \pi_i = k) \tag{3.20}$$

$$= \sum_k P(x_{i+1} | \pi_{i+1} = k) b_k(i+1) P(\pi_{i+1} = k | \pi_i = l) \tag{3.21}$$

The computation follows in the backward direction.

i	$\ldots$	16	17	$\ldots$
x	$\ldots$	$x_{16} = K$	$x_{17} = F$	$\ldots$
$b_+(i)$	$\ldots$	$\overleftarrow{\swarrow} b_+(16) = \begin{pmatrix} P(F\|+)b_+(17)P(+\|+) \\ + P(F\|-)b_-(17)P(-\|+) \end{pmatrix}$	$\overleftarrow{\swarrow} b_+(17)$	$\ldots$
$b_-(i)$	$\ldots$	$\overleftarrow{\nwarrow} b_-(16) = \begin{pmatrix} P(F\|+)b_+(17)P(+\|-) \\ + P(F\|-)b_-(17)P(-\|-) \end{pmatrix}$	$\overleftarrow{\nwarrow} b_-(17)$	$\ldots$

Similarly, the result of the backward computation at the *start* of the sequence is the probability of the sequence.

$$P(x) = \sum_k P(start | x_1) P(x_1 | \pi_1 = k) b_k(1) \tag{3.22}$$

Most Probable State Given the sequence, we may be concerned about the most probable state of a particular amino acid at a certain position. Is that amino acid functionally important or not? This may be different from the states given by the most probable state path as calculated by the Viterbi algorithm above. We want to know the probability of having a certain state k at position i given the entire sequence x. It turns out that this can be determined by using the calculations from the forward and backward algorithms.

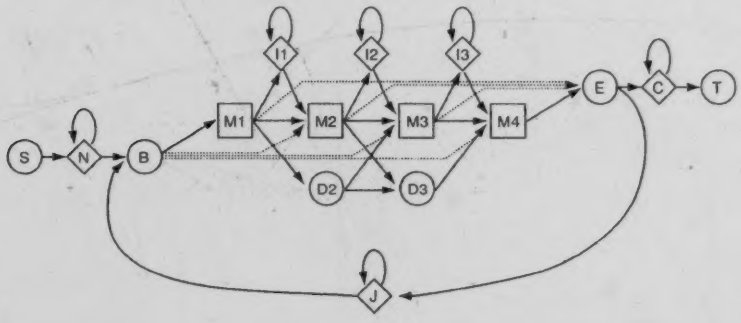
$$P(\pi_i = k | x) = \frac{P(x_1, \ldots, x_i, \pi_i = k) P(x_{i+1}, \ldots, x_L | \pi_i = k)}{P(x)} \tag{3.23}$$

$$= \frac{f_k(i) b_k(i)}{P(x)} \tag{3.24}$$

We can use the values obtained from forward and backward sweeps through the sequence to calculate the most probable state at each position in the sequence.

Figure 3.16: HMMer Architecture.

This model structure of a profile HMM incorporates several types of hidden states. Diamonds indicate insert (I) states, circles indicate delete (D) states, squares indicate match (M) states.



i	...	16	17	18	19	20
x	...	K .	F	I	N	...
$f_+(i)$	...	$f_+(16)$	$f_+(17)$	$f_+(18)$	$f_+(19)$	...
$f_-(i)$	...	$f_-(16)$	$f_-(17)$	$f_-(18)$	$f_-(19)$	...
$b_+(i)$	...	$b_+(16)$	$b_+(17)$	$b_+(18)$	$b_+(19)$	...
$b_-(i)$	...	$b_-(16)$	$b_-(17)$	$b_-(18)$	$b_-(19)$	...
$P(\pi_i = +\|x)$	...	$\frac{f_+(16)b_+(16)}{P(X)}$	$\frac{f_+(17)b_+(17)}{P(X)}$	$\frac{f_+(18)b_+(18)}{P(X)}$	$\frac{f_+(19)b_+(19)}{P(X)}$	...
$max_k P(\pi_k\|x)$	...	↓	↑	↑	↑	...
$P(\pi_i = -\|x)$	...	$\frac{f_-(16)b_-(16)}{P(X)}$	$\frac{f_-(17)b_-(17)}{P(X)}$	$\frac{f_-(18)b_-(18)}{P(X)}$	$\frac{f_-(19)b_-(19)}{P(X)}$	...
MPS	...	$-$	$+$	$+$	$+$	...

Note that the states obtained by this example calculation are different than those obtained for the most probable state path through the Viterbi algorithm.

HMMer HMMer [35] is an implementation of *profile* hidden Markov models [34]. Profile HMMs are HMMs that have been designed to represent protein sequence families. These models take positional information along the sequence into account in a manner analogous to PSSMs, but the power of hidden states also allows them to deal with patterns of flexible lengths (through deletion and insertion states that are analogous to deletions and insertions in sequence alignments). The parameters of each model are typically built up from a multiple sequence alignment of the family of sequences.

The model structure of the profile HMMs used by HMMer is seen in Figure 3.16. The M states are *match* states. The I states are *insertion* states. Match states and insertion states both have associated emission probabilities for each amino acid. The D states are *deletion* states that do not emit symbols. Since deletion states are 'silent', a sequence might correspond to a path through the model which may essentially skip an amino acid by passing through a deletion state instead of a match state. The length of the center portion of the model (including M, I, and D states) is variable and is built to reflect the typical length of the sequence domain or family being modeled.

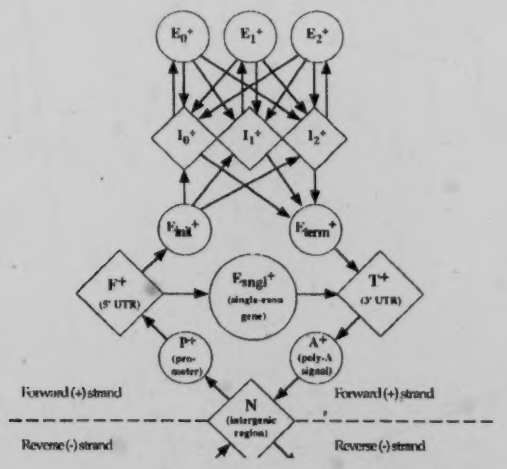
Consider how the pattern 'FIND' in the example sequences might be modeled using the profile HMM structure shown in Figure 3.16. As this pattern is of length 4, each letter in the pattern might correspond to match states in the model — 'F' to $M1$, 'I' to $M2$, 'N' to $M3$, and 'D' to $M4$. Given training data that contains slightly noisy variations of the pattern 'FIND', each of these states from $M1$ to $M4$ may emit any amino acid – but they each will emit the residues matching the pattern with the highest probability. The sequence 'FIND' will be assigned the highest possible probability given a model that has been trained on sequences containing this pattern. The most probable state path traced through the model in matching 'FIND' will pass directly through the 4 match (M) states. A sequence 'FUND' may still match the pattern, but with a lower probability than 'FIND'. Insertion states also emit amino acids. When insertions are included, the sequence 'FOIND' might also match the pattern with the 'O' matching state $I1$, although this would also have a lower probability than 'FIND'. A sequence may also have a path through the HMM that passes through the silent delete states. The sequence 'FID' may match the model with a path through the delete state $D3$, again with a lower probability than the perfect match 'FIND'. Thus, the match, insertion, and deletion states in profile HMMs provides much greater flexibility than that afforded by PSSMs or exact matching models.

HMMer is one of the most commonly used HMM tools for protein families and is the basis for the Pfam database of protein families (see below). Despite their advantages over other techniques such as exact regular expressions and PSSMs, HMMs also have some drawbacks. Profile HMMs typically require a very computationally expensive multiple sequence alignment. In addition, they are not designed to represent large heterogeneous families of sequences.

Other HMM Predictors Other HMM predictors have been used for a variety of very specific tasks. Among the many examples, TMHMM [88] is used to predict transmembrane helices and SignalP-HMM [77] is used to predict signal peptide cleavage sites. GENSCAN [24] uses an HMM to predict gene regions in DNA. Model architectures can vary widely by task and are often designed to mirror the biological structure of the sequences being modeled. Figure 3.17 shows part of

Figure 3.17: GENSCAN HMM Architecture.

GENSCAN is used to identify genes in eukaryotic DNA. The states in the model reflect biological realities and complexities in the sequence. The E states represent exons, the part of a gene that is 'expressed' and later translated to amino acids. The I states represent introns which occur between the exons.



the HMM architecture of GENSCAN that is designed to model genes in DNA sequences. The E states model exons and the I states model introns. The triple intron and exon states at the top of the diagram reflect the nature of DNA codon triplets. Although it is unnecessary to understand the details of this structure for our discussion, it is interesting to note the biological realities and complexities which may be reflected in hidden Markov models. As the model structure in each case is designed specifically for the intended prediction task, the performance of these HMMs tends to be better than using the more generic program (such as HMMer).

PFAM Pfam [13] is a database of profile hidden Markov models [34] and their associated multiple alignments for a large number of protein families (over 7000). These profile HMMs have a relatively restricted architecture (as defined by HMMer and seen in Figure 3.16) that has been developed to represent a wide variety of protein families. Pfam contains HMMs that can model either entire sequences or smaller domains and motifs. The database can be searched using the HMMer [35] software.

As an example of a Pfam entry, the Pfam profile HMM `PF00516` models the GP120 envelope glycoproteins of the HIV virus that causes AIDS. The GP120 envelope proteins are responsible for attachment of the HIV virus to human white blood

Figure 3.18: Pfam Domains of the HIV Protein GP120

Figure 3.19: Structure of the HIV Protein GP120

cells. The profile HMM probabilistically describes the two main larger domains of the proteins and the few smaller domains (shown by rectangles of various sizes along the sequence in Figure 3.18). A protein that has a sequence that matches the profile HMM for the GP120 family is very likely to also have a three-dimensional structure that is similar to that of the GP120 family (seen in Figure 3.19). A new protein sequence with functionality similar to that of GP120 might be recognized by a high-scoring match (low E-value) to one or more of these domains.

Figure 3.20 displays some example Pfam results as run on the third and fourth sequences of the example set.

Although Pfam families have been designed for specific tasks, they may also be of some use for situations in which no single Pfam family exactly matches our protein family of interest. Given that building custom classifiers for each family can be expensive, we may wish to use well-established protein families that have some relation to the protein class we wish to identify and classify new proteins based on some combination of the patterns representing that family.

3.4 Kernel Methods

Support vector machines (see Section 2.4.4) allow interesting ways of comparing sequences that may make use of some of the sequence patterns above. By defining a string kernel, we can classify sequences using complex patterns in the sequence data. Although simple information such as common substrings or substring frequencies might be used by a kernel, it may represent almost any arbitrary measure of the similarity between two sequences (subject to the constraints mentioned in Section 2.4.4). This kernel may be optimized for classification performance for a specific protein function. Algorithmic methods also allow very efficient computation for some tasks. Two examples of kernels that have been defined for sequence analysis include the Fisher kernel for HMMs [47] and mismatch string kernels [57]. Although current string kernel methods are relatively inefficient for high-throughput prediction, this is a very promising area of future research.

3.5 Summary

There is a wide variety of pattern types that may be used for predicting protein function. Both deterministic and probabilistic methods may be used. Patterns vary in their classification performance and computational requirements. There is often a trade-off between sensitivity and computational requirements. The computational requirements are also very different for training and classification. We would ideally like to utilize a technique that is robust and efficient for both training and testing.

Figure 3.20: Example Pfam Results

```
Query sequence: 3
Accession:      [none]
Description:    [none]

Scores for sequence family classification (score includes all domains):
Model         Description                                    Score    E-value  N
--------      -----------                                    -----    ------- ---
14-3-3        14-3-3 protein                                 191.2    1.4e-54   3
PolC_DP2      DNA polymerase II large subunit DP2               2.2     0.0085   1
CO_dh         CO dehydrogenase beta subunit/acetyl-Co           1.7        1.7   1
CAT_RBD       CAT RNA binding domain                            3.1        1.9   1
Orbi_VP5      Orbivirus outer capsid protein VP5               -0.2        2.2   1
Ribosomal_S14 Ribosomal protein S14p/S29e                       0.2        7.4   1

Parsed for domains:
Model         Domain  seq-f seq-t    hmm-f hmm-t      score  E-value
--------      ------- ----- -----    ----- -----      -----  -------
14-3-3          1/3       1    12 [.     8    19 ..     3.5     0.33
14-3-3          2/3      13    76 ..    40   105 ..   106.7  3.8e-30
Ribosomal_S14   1/1      18    30 ..    89   102 .]     0.2      7.4
Orbi_VP5        1/1      41    59 ..   176   194 ..    -0.2      2.2
PolC_DP2        1/1      51    64 ..     1    14 [.     2.2   0.0085
14-3-3          3/3      92   152 .]   102   163 ..    81.1  5.5e-23
CAT_RBD         1/1     105   118 ..    49    62 .]     3.1      1.9
CO_dh           1/1     114   123 ..   171   180 .]     1.7      1.7
//

Query sequence: 4
Accession:      [none]
Description:    [none]

Scores for sequence family classification (score includes all domains):
Model         Description                                    Score    E-value  N
--------      -----------                                    -----    ------- ---
GTP1_OBG      GTP1/OBG family                                170.0    3.5e-48   3
ArgK          ArgK protein                                     9.7     0.0068   1
MMR_HSR1      GTPase of unknown function                       6.0       0.13   1
FeoB          Ferrous iron transport protein B                 3.6       0.16   1
DnaB_C        DnaB-like helicase C terminal domain            -0.2        2.9   1
ABC_tran      ABC transporter                                  0.8          4   1
FrhB_FdhB_C   Coenzyme F420 hydrogenase/dehydrogenase,        -0.2        5.9   1
MCR_beta_N    Methyl-coenzyme M reductase beta subunit,       -0.9        6.8   1
PGK           Phosphoglycerate kinase                         -1.4        7.9   1

Parsed for domains:
Model         Domain  seq-f seq-t    hmm-f hmm-t      score  E-value
--------      ------- ----- -----    ----- -----      -----  -------
GTP1_OBG        1/3       2    29 ..     1    28 [.    35.7  6.6e-10
GTP1_OBG        2/3      30    68 ..    51    92 ..    42.1  1.2e-11
ArgK            1/1      41    65 ..    28    53 ..     9.7   0.0068
FrhB_FdhB_C     1/1      44    54 ..     1    11 [.    -0.2      5.9
ABC_tran        1/1      45    67 ..     1    23 [.     0.8        4
MMR_HSR1        1/1      47    62 ..   133   148 ..     6.0     0.13
FeoB            1/1      50    67 ..     1    18 [.     3.6     0.16
DnaB_C          1/1      54    66 ..    31    43 ..    -0.2      2.9
PGK             1/1      62    73 ..   161   172 ..    -1.4      7.9
GTP1_OBG        3/3      88   142 .]    93   149 ..    92.2  1.4e-25
MCR_beta_N      1/1     106   124 ..   167   186 .]    -0.9      6.8
//
```

Chapter 4

Probabilistic Suffix Trees

'A tree's a tree. How many more do you need to look at?'[1]

The following chapter will introduce suffix trees, their construction, and their use. Two implementations of probabilistic suffix trees will then be presented with some discussion of their properties.

4.1 Suffix Trees

Suffix trees are data structures that represent strings in a way that allows very efficient manipulation and analysis. By representing all the suffixes of a string in a tree, the internal structure of the string can be quickly analyzed. Much of the information in this section is adapted from Gusfield [40], where readers may find a complete discussion of suffix trees and their variations.

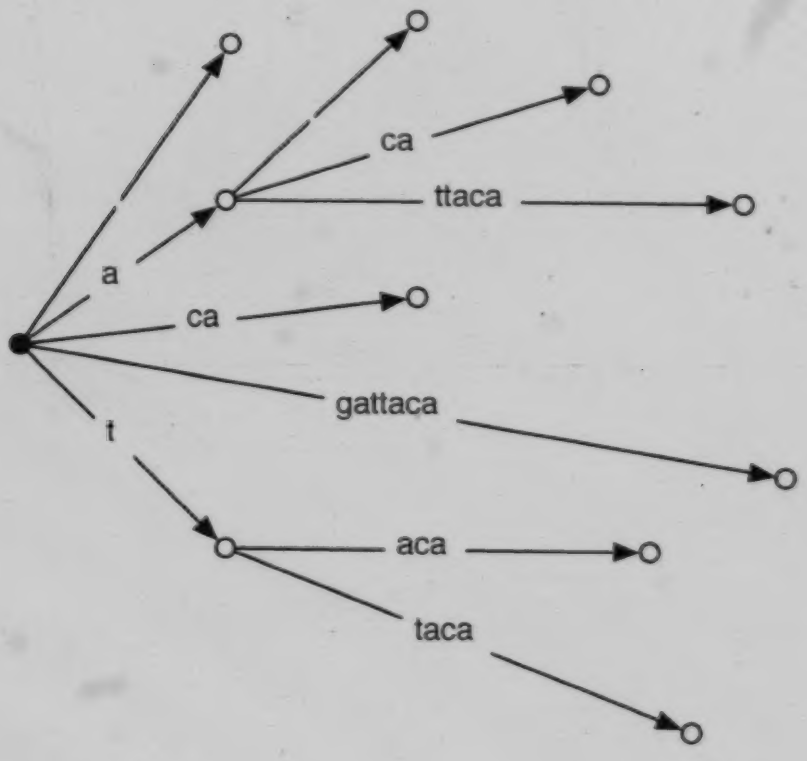
The suffixes of the word 'gattaca' [76] are 'gattaca', 'attaca', 'ttaca', 'taca', 'aca', 'ca', 'a', and ''. Notice that the string itself and the empty string are both considered suffixes in this context. Each suffix is represented in the suffix tree by a path from the root to a leaf (see Figure 4.1). The unique path for each suffix is labeled with the letters of the suffix. Note that each edge exiting a node must begin with a different symbol than all other edges leaving that node.

This data structure allows us to quickly answer questions about the sequence represented by the suffix tree. We might ask, 'Is the query string "aca" a substring of "gattaca"?' A naïve method of answering this question would scan along the word 'gattaca' for a match to the first letter 'a' of the query. When an 'a' is found, we might check for a match to the remainder of the letters in 'aca'. We would find that the first match to 'a' is followed by a 't', so we must continue our scan. At the second 'a' we find a match that allows us to match the complete 'aca'. In total, we would use 8 comparisons to establish that 'aca' is found in 'gattaca'. By using a

[1] Ronald Reagan, U.S. President, in a speech delivered on Sept. 12, 1965 and quoted in the Sacramento Bee (California, Mar. 12, 1966). Reagan later denied having made this statement.

Figure 4.1: A Suffix Tree

A suffix tree representing the word 'gattaca'. Each suffix of the word can be traced along edges from the root node (filled with black) to a leaf node. The edges without labels represent the empty string.



suffix tree, we can follow the path from the root along the path marked 'aca'. Since there is such a path in the suffix tree, we know that 'aca' is in at least one of the suffixes of 'gattaca' and thus must be in the word itself. Following this path by checking each edge alphabetically for a match in the example tree would require 5 comparisons. The time required to check for 'aca' in the naïve way increases with the length m of the target string 'gattaca' and length n of the query string 'aca' ($O(nm)$ in the worst case). The time required to check by using a suffix tree increases linearly with respect to the length of the query string ($O(n)$). Although the savings made by using a suffix tree are small for this simple example, they become very large over longer strings such as DNA or protein sequences.

4.1.1 Building and Using Suffix Trees

Let us consider a protein x consisting of a sequence of amino acids ($x_1 \ldots x_m\$$ – here the symbol \$ indicates the end of the sequence). Assume that there are only two symbols in this protein alphabet $\{A, Y\}$ with 'A' representing all hydrophobic amino acids and 'Y' representing all hydrophilic amino acids. Given a sequence 'YAYYAYA', we would like to construct a suffix tree T. One way to construct a suffix tree for the sequence is by inserting each suffix into the tree in turn, from the longest ($x_1 \ldots x_m\$$) to the shortest (\$ – the empty string). With the insertion of each string, we follow edges where characters are already in place and create new nodes and edges where needed (see Figure 4.2). The time required by this procedure grows quadratically with the length of the sequence $O(m^2)$.

After construction of the tree we can perform many types of analysis efficiently. We can quickly determine the number of occurrences of the string 'YA' in the sequence by following the path marked 'YA' and then counting the number of leaf nodes below that point (three leaf nodes, as seen in Figure 4.3). We can also quickly determine the locations of each 'YA' occurrence since the distance from the 'YA' node to the leaf nodes equals the distance from each occurrence to the end of the sequence.

The basic suffix tree algorithms can be extended in many ways (as demonstrated by Gusfield [40]). Multiple strings can be contained within a single suffix tree. The data structure is then called a *generalized suffix tree* (see Figure 4.4). This structure leads to solutions to common substring problems. We can, for example, very quickly determine the longest common substring in a set of sequences by finding the deepest node with leaves representing each sequence in the set. We can also use relatively few comparisons to count the number of occurrences of all the substrings in a set of sequences in $O(m)$ time (also see Figure 4.4). This ability will be key to efficiently calculating conditional probabilities for variable-length Markov chains (see Section 4.3).

Figure 4.2: Building a Suffix Tree

Building a suffix tree representing the sequence 'YAYYAYA' by adding each suffix sequentially from longest to shortest. The symbol $ indicates the end of the sequence. The black filled node is the root.

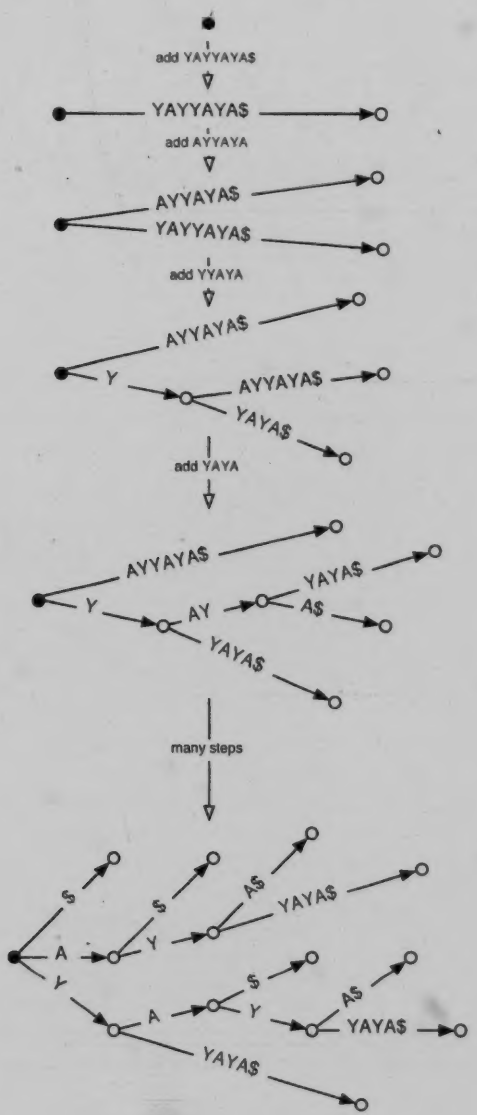


Figure 4.3: Counting Substring Occurrences with a Suffix Tree

The suffix tree representing the sequence 'YAYYAYA' can be used to count the number of occurrences of the substring 'YA' by following the path (indicated in bold) from the root and then counting the number of leaf nodes below the point reached in the tree. The $ symbol indicates the end of the sequence.

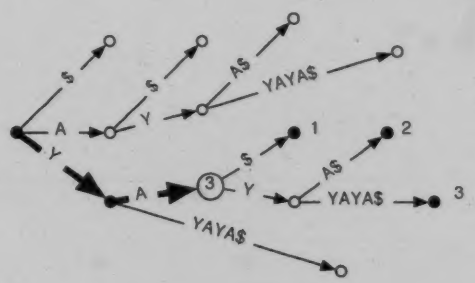


Figure 4.4: A Generalized Suffix Tree

The generalized suffix tree representing the sequences 'YAYYAYA' and 'AYYAYY' with the unique symbols '$' and '@' terminating the respective sequences. The root node is filled with black. Each node contains the number of leaf nodes at or below that node. This indicates the number of occurrences of the string represented by the path from the root. These numbers can be calculated efficiently with a depth-first traversal of the tree.

4.1.2 Linear Time Construction of Suffix Trees

Although we can see above that suffix trees allow very efficient operations over sequences, we have neglected to include the cost of creating the tree in our performance analysis. The $O(m^2)$ algorithm for suffix tree construction shown above requires significant computational resources and would prevent our use of suffix trees for many applications. It is fortunate — and somewhat surprising — that suffix trees can be built in linear time $O(m)$ with respect to the length m of the sequence[2]. An algorithm for building suffix trees in linear time was first presented by Weiner [97] in 1973[3], followed by a more space-efficient algorithm from McCreight [68] in 1976. Almost 20 years later, Ukkonen [94] published a version of the algorithm that, although equivalent to the algorithms of McCreight and Weiner in time complexity, has some distinct advantages. The main advantage is that it is much simpler to understand. Ukkonen's algorithm is fully developed elsewhere and we will focus on only a few highlights that are particularly relevant to this work. For a complete discussion of linear time construction of suffix trees, interested readers should refer to Gusfield [40].

The following three details of Ukkonen's algorithm (as elucidated by Gusfield) will be important for our later discussion of efficient probabilistic suffix trees.

Online construction Ukkonen's algorithm builds the tree in order from the beginning to the end of the input sequence $X = x_1 \ldots x_m$. After each character x_i is processed, the tree T contains the entire set of suffixes for the sequence $x_1 \ldots x_i$. There is no requirement of having the entire sequence $x_1 \ldots x_i \ldots x_m$ before processing (as in McCreight's algorithm) or even knowing the length m of the sequence beforehand.

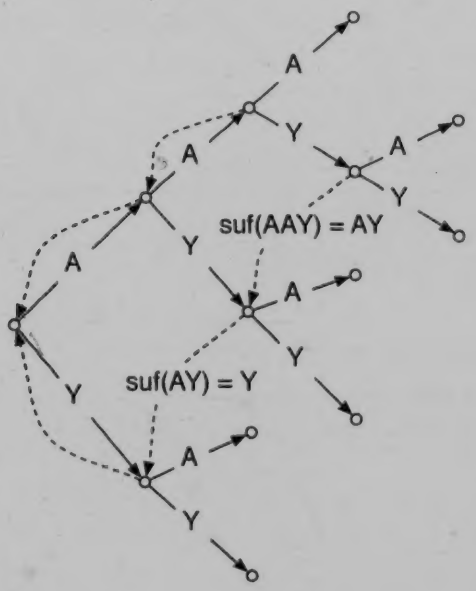
Edge-label compression An edge between nodes in the suffix tree may represent one or more symbols (as shown in the Figures 4.1 and 4.2). To allow the linear time algorithm, these symbols along the edges are represented by a pointer to the symbols in the original sequence. This is necessary because the storing copies of each symbol in the tree would require a quadratic increase $O(m^2)$ in the space needed to store the tree. A quadratic increase in space would immediately negate the possibility of a linear time algorithm as a quadratic amount of data cannot be copied in a linear amount of time. Because of edge-label compression, the traversal of an edge can be done in one step (constant time $O(1)$) regardless of length.

[2]We acknowledge that the time and space bounds for this algorithm (and the many subsequent suffix tree algorithms presented here) are actually dependent on the size of the alphabet Σ such that the time bound becomes $O(m|\Sigma|)$. Since the size of the protein alphabets will be constant for protein function prediction problems we will ignore the contribution of this component.

[3]Knuth is claimed to have called this '*the* algorithm of 1973' [40, p.90].

Figure 4.5: Suffix Links

The dotted lines represent suffix links in this partial suffix tree.



Suffix links The immediate suffix $suf(\cdot)$ of a string 'Xs' is that same string without the first symbol X.

$$suf(Xs) = s \tag{4.1}$$

Suffix links are pointers from each node in a suffix tree representing a string s to the node representing the immediate suffix $suf(s)$ of that string. Each internal node (non-leaf) of the suffix tree has a suffix link to the node that represents its immediate suffix. Figure 4.5 illustrates how suffix links appear in the tree. For internal nodes that represent only a single symbol, the suffix link will be to the root node which represents the empty string. Suffix links allow one step traversal of the tree from any string represented by a node to its immediate suffix.

Although the details will not be presented here, it is also possible to jump from a path that ends in the middle of an edge to its immediate suffix by using the nearest suffix link above that position. Gusfield calls this the *skip/count trick*. Although the skip/count trick is slightly more computationally expensive than simply following a suffix link, it does not affect the overall linear time bound.

A full description of Ukkonen's algorithm as well as additional details required

for the linear time construction of suffix trees is developed by Gusfield [40].

4.2 Probabilistic Suffix Trees

4.2.1 Variable Length Markov Models (VMMs)

The principles that apply to the use of first-order Markov chains for classification (discussed in Section 3.3.2) may be extended to higher-order Markov chains. As a k-order Markov chain requires a history of length k for each conditional probability, the storage requirements of a complete higher-order Markov chain increase exponentially with the order k. This exponential memory requirement is prohibitive for many practical applications.

The space requirements are not the only obstacle to using higher-order Markov chains. As the order increases, so does the amount of training data needed to give good parameter estimates. Given a k-order Markov chain we might estimate the probability of a symbol x_i given its history $x_{i-1}, \ldots, x_{i-k}$ by counting the occurrences ($C(\cdot)$) over the training data (maximum likelihood estimation).

$$\tilde{P}(x_i|x_{i-k}\ldots x_{i-1}) = \frac{C(x_{i-k}\ldots x_{i-1}x_i)}{C(x_{i-k}\ldots x_{i-1})} \tag{4.2}$$

With smaller counts the estimated probabilities are less likely to accurately represent the population of sequences that might be observed. Counts of zero also present problems. If there are *no* occurrences of the sequence $x_{i-k}\ldots x_{i-1}x_i$ in the data then the assigned probability will be zero, which cannot reflect the true distribution if an occurrence of $x_{i-k}\ldots x_{i-1}x_i$ is possible in the test set. An even worse case may occur if there are no occurrences of the sequence $x_{i-k}\ldots x_{i-1}$ in the training sequences because the probability calculation then involves division by zero and is undefined. With increasing history length, the training data becomes more sparse in the model and parameter estimation problems become increasingly prevalent.

Variable length Markov Models (VMMs) have been proposed to deal with both space and parameter-estimation limitations by utilizing longer or shorter histories for prediction as needed. This flexibility helps deal with the space problems encountered with complete higher-order Markov models because only the parameters that are needed are stored. VMMs can also avoid some of the problems associated with lack of training data. For example, where sufficient training data is not available for a higher-order prediction, the model may approximate that value by using a lower-order parameter. The criteria used for selecting Markov order, or *context*, is not necessarily restricted to the availability of data. The context selection algorithm

may be based on any criteria that may improve the classification performance of the model.

After a VMM is created from a set of training sequences it may be used to predict the chance that a test sequence belongs to the family of training sequences. The predictions obtained from a VMM are very similar to those obtained from a simple Markov chain (see Section 3.3.2). The probability of the entire sequence is calculated as the product of the probabilities of each amino acid given those that precede it.

$$\begin{aligned} P(x) &= P(x_1 \dots x_m) && (4.3) \\ &= P(x_1)P(x_2|x_1)P(x_3|x_1x_2)\dots \\ &\quad P(x_{m-1}|x_1 \dots x_{m-2})P(x_m|x_1 \dots x_{m-1}) && (4.4) \\ &= P_{VMM}(x_1)P_{VMM}(x_2|x_1)P_{VMM}(x_3|x_1x_2)\dots \\ &\quad P_{VMM}(x_{m-1}|x_1 \dots x_{m-2})P_{VMM}(x_m|x_1 \dots x_{m-1}) && (4.5) \end{aligned}$$

Instead of obtaining a probability that is dependent on a fixed history, the VMM selects which conditional probability to use based on the context of the amino acid. An arbitrary context function $\kappa(\cdot)$ can select the history length based on the context $x_1 \dots x_{i-1}x_i$.

$$P_{VMM}(x_i|x_1 \dots x_{i-1}) = \begin{cases} \tilde{P}(x_i|x_1 \dots x_{i-1}) & \text{if } \kappa(x_1 \dots x_i) = i \\ \tilde{P}(x_i|x_2 \dots x_{i-1}) & \text{if } \kappa(x_1 \dots x_i) = i-1 \\ \vdots & \vdots \\ \tilde{P}(x_i|x_{i-2}x_{i-1}) & \text{if } \kappa(x_1 \dots x_i) = 2 \\ \tilde{P}(x_i|x_{i-1}) & \text{if } \kappa(x_1 \dots x_i) = 1 \\ \tilde{P}(x_i) & \text{if } \kappa(x_1 \dots x_i) = 0 \end{cases} \quad (4.6)$$

The idea of selecting context is similar to the idea of *backoff* as used in Markov chains for natural language processing [50]. In general, it is not necessary to select a single conditional probability. Some weighted combination of the parameters may be used. Related concepts that have been used to deal with parameter estimation issues in Markov chains [50] include *smoothing* and *deleted interpolation*. We will address the topic of smoothing later.

The literature on VMMs originated in information theory and data compression in 1983 [81]. The principles have been used in natural language applications with variable length N-grams [75]. VMMs were introduced to the computational biology literature in the form of probabilistic suffix trees (PSTs). Various statistical properties of VMMs (also known as Variable Length Markov Chains - VLMCs) and some context selection techniques have also been examined [23]. There has been work to

generalize probabilistic suffix trees through sparse Markov transducers [37] which allow wildcards in the training set.

4.2.2 Probabilistic Suffix Trees

Probabilistic suffix trees (PSTs) were introduced by Ron, Singer, and Tishby [82]. Bejerano and Yona [16, 17] further developed this work and brought it to the molecular biology community. PSTs (also known as prediction suffix trees) are variable length Markov models (VMMs) that are designed for efficient prediction. As VMMs, they avoid the exponential space requirements and parameter estimation problems of higher-order Markov chains. A PST is pruned during training to contain only the required parameters.

An example PST fragment as described by Bejerano [15] is seen in Figure 4.6. This PST was created and pruned using the training sequence '0100100100111101-01100010111'. As the following discussion will refer to the PSTs presented by Bejerano and Yona [16] (and implemented by Bejerano [15]), we will refer to the models as bPSTs.

bPSTs differ from classical suffix trees (as discussed in Section 4.1) in that they are not used to represent the training sequences. bPSTs instead represent the *histories* used for prediction in the tree. Since the histories are represented with the rightmost symbols (reading the symbols in the sequence from left to right) closer to the root, the tree is drawn with the root on the right. An example of how to read the history '001' from the tree is illustrated in Figure 4.6.

We will discuss first how bPSTs are used in prediction and then how they are constructed.

Prediction with a bPST

The conditional probability $P_{bPST}(x_i|x_1 \ldots x_{i-1})$ is obtained from the bPST by tracing a path from the root that matches the amino acid residues preceding the current residue. The parameters γ that estimate the conditional probabilities are stored at each node in the bPST.

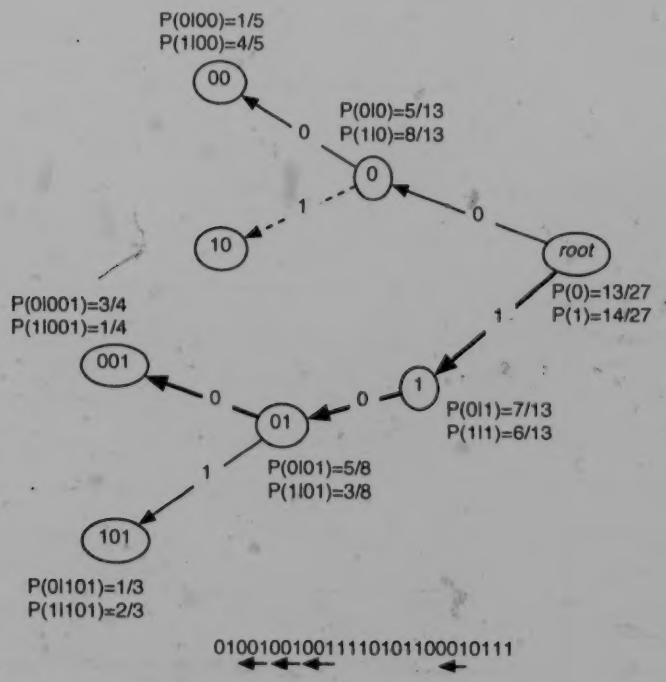
$$\gamma_{history}(a) \approx P(a|history) \tag{4.7}$$

During prediction, the longest available context that matches the history of the current amino acid is used.

$$P_{bPST}(x_i|x_{i-1}, \ldots, x_1) = \tilde{\gamma}_{x_1 \ldots x_{i-1}}(x_i) \tag{4.8}$$

Figure 4.6: Bejerano and Yona's Probabilistic Suffix Tree

This example bPST fragment is reproduced from Bejerano [15] with modifications for clarity. Note that the root is represented on the right (in contrast to the left for classical suffix trees presented earlier). The path to the node representing the history '001' is shown in **bold**. The corresponding subsequences in the training sequence are noted by arrows. Note that of the 4 occurrences, 3 are followed by a '0' and 1 is followed by a '1', giving rise to the calculated probabilities shown at node '001'.



$$= \begin{cases} \gamma_{x_1 \ldots x_{i-1}}(x_i) & \text{if in } bPST, \text{ else} \\ \gamma_{x_2 \ldots x_{i-1}}(x_i) & \text{if in } bPST, \text{ else} \\ \vdots & \vdots \\ \gamma_{x_{i-2} x_{i-1}}(x_i) & \text{if in } bPST, \text{ else} \\ \gamma_{x_{i-1}}(x_i) & \text{if in } bPST, \text{ else} \\ \gamma(x_i) & \end{cases} \tag{4.9}$$

The longest history can be found quickly by tracing back through the PST along a path which matches the amino acids immediately preceding the current residue. For example, given the bPST in Figure 4.6 and the sequence '01001' we can obtain the predictions for each amino acid by tracing back through the tree. The underlined characters in the calculation below indicate those characters that will be used as the history for each prediction. The predictions marked γ^* use truncated histories because the full history was not found in the tree.

$$\begin{aligned} P(01001) &= P(\underline{0})P(1|\underline{0})P(0|\underline{01})P(0|01\underline{0})P(1|01\underline{00}) \\ &= \gamma(0)\gamma_0(1)\gamma_{01}(0)\gamma_0^*(0)\gamma_{00}^*(1) \\ &= (13/27)(8/13)(5/8)(5/13)(4/5) \\ &= 10400/182520 = 0.057 \end{aligned}$$

In this way, each prediction for the conditional probability of each residue can be done in linear time $O(L)$ with respect to the maximum history length L of the model. The worst-case time taken by the entire prediction increases with the length of the sequence m and the history length (order) L of the model $O(mL)$.

The global prediction of the probability of the entire sequence is useful for many tasks. Often, however, we have biological reasons to believe that the regions which are important for defining the protein class do not span the entire sequence. For this reason, local predictions over some shorter sequence length may be beneficial. Local predictions using PSTs have been done in a manner identical to that used for Markov chains (see Section 3.3.2). The product of the prediction probabilities is calculated for a fixed-length window. The product for the sliding window is calculated for each point in the sequence. Sun and Deogun [89] have worked with some other methods of local prediction that extend the sliding window approach. They automatically select the best window size over a range of potential sizes and obtain modest improvements in classification performance.

Ron *et al.* [82] proposed a method that allows for linear time prediction using the information from a bPST by building an equivalent Probabilistic Finite Automaton (PFA). The conversion to a PFA from a bPST costs $O(Lm^2)$ time where m is the total combined length of the training set.

Building a bPST

The bPST is built by progressively adding the histories from the training data to the tree [16]. All the substrings of the training sequences are possible histories. The histories that are added to the tree must be smaller than the maximum history L of the tree and must occur more frequently than some threshold P_{min}. The substrings are added in order of length from smallest to largest. The histories are added to the tree in reverse (with the rightmost symbols of the subsequence added first starting at the root). The possible histories are the reversed strings of all the substrings of the training sequences. The string s is only added to the tree if the resulting conditional probability (of a symbol given its history) at the node to be created will be greater than the minimum prediction probability $\gamma_{min} + \alpha$ and the probability for the prefix of the string is different (with some ratio r) from the probability assigned to the next shortest substring $suf(s)$ (that is already in the tree). After all the substrings are added to the tree, the probabilities are smoothed according to the parameter $\gamma_{min} \in [0, 1]$. The smoothing probability $\gamma_{history}(a)$ for a symbol a given its $history$ (as calculated by the equation below) prevents any probability from being less than γ_{min}. Σ is the size of the alphabet and $\tilde{P}$ is a probability estimated from occurrences in the training set.

$$\gamma_{history}(a) = (1 - |\Sigma|\gamma_{min})\tilde{P}(a|history) + \gamma_{min} \tag{4.10}$$

This building process requires $O(Lm^2)$ time [7], where L is the maximum history length of the tree and m is the total combined length of the training set. The building process requires the all the training sequences at once (in order to get all the reverse substrings) and cannot be done *online* (the bPST cannot be built incrementally as the training data is encountered).

Bejerano and Yona [16] also suggested an alternate smoothing technique which utilizes pseudocounts that have been chosen based on the amino acid frequencies of well-studied protein databases. In their experiments, the performance improvements due to this smoothing were minimal. Other authors [51], however, have used smoothing and backoff methods similar to those used for natural language processing [50] to some advantage.

4.2.3 PSTs vs HMMs

PSTs have been compared to HMMs and BLAST as predictors of protein families [16, 51, 89]. The classification performance has not been shown to exceed HMMs for classification on the Pfam or SCOP databases, but the results have been very encouraging. Although PSTs are not as flexible and powerful as HMMs in their representation of sequence classes, they have been shown to perform as well as

HMMs for many protein families. The incremental improvements which have been suggested in smoothing [51] and local scoring [89] have improved the results significantly. Further evaluation of PSTs using a combination of the previously suggested improvements may yield results that exceed HMM classification performance for some problems. The main advantage of PSTs over HMMs is that the training and prediction time requirements of PSTs are much less than for the equivalent HMMs (see Section 5.5). A large part of the computational cost of HMMs is due to the requirement of a multiple sequence alignment. These multiple alignments also often require further tuning by researchers – an additional resource cost. PSTs are completely alignment free and do not require expert tuning. It should also be noted that when the expectation maximization (EM) algorithm is used to train an HMM there is a danger of the parameters reaching a poor local minimum. Overall, the reduced resource demands of PSTs make them an attractive alternative to HMMs despite a potential loss in classification performance.

4.3 Linear Time Construction and Prediction with PSTs

Shortly after the initial publication of bPST results, Apostolico and Bejerano [7] presented theoretical results for linear time construction of suffix trees and linear time prediction using suffix trees. These theoretical results support the construction of the tree in linear time $O(m)$ with respect to the total combined length of the training sequences m. They also present a method of prediction that is linear $O(n)$ in time with respect to the length of the sequence being tested. The method involves building a suffix tree to represent the training sequences (in contrast to a tree that represents the histories, as bPSTs do). This suffix tree is then used to efficiently count the number of occurrences of the subsequences of the training set. The tree may then be pruned by the same criteria used by Bejerano and Yona in their previous work [16], although the pruning algorithm differs. The resulting suffix tree may be used for linear time prediction by using the suffix links as failure points. The resulting model structure is similar to Aho and Corasick's prior work with Multiple Pattern Matching Machines (MPMMs) [2].

Despite the statement at the conclusion of Apostolico and Bejerano's paper [7] that, '...one of the main hinges along the way of a computational tool to become practicable by the bioinformatics community is its run time requirements,' there is no evidence of an implementation of this algorithm. Indeed, Bejerano's recent public release of his PST implementation [15] contains only the quadratic time algorithms of the earlier papers. No results for an implementation of this algorithm have been available until recently [89], coincident with the development of this work. The authors of that paper do not, however, made any implementation publicly

available. The training time of that implementation is noted to be about 180 times faster than Bejerano's. No comparison of prediction performance is mentioned.

With high-throughput protein function prediction as a motivation, the theoretical efficiencies of PSTs are very attractive. The linear time $O(m)$ training and linear time prediction $O(n)$ characteristics would make the use of PSTs practical for a much larger range of protein function prediction tasks. The current work offers a presentation of the linear time training and prediction algorithms. This presentation is in many ways equivalent to the algorithm of Apostolico and Bejerano [7]. Differences in approach are due to independent development and slightly different research motivation. Various improvements to the technique will also be touched upon. The presentation leads to an implementation of the algorithm and the associated results (see Chapter 5).

Bejerano's recently published [15] PST implementation (bPSTs) does not take advantage of the full potential of suffix trees to efficiently train and predict VMMs. The current presentation of PSTs (which we will call *efficient* PSTs or *ePSTs*) will illustrate the linear time construction and use of PSTs for protein function prediction. Other notes and variations on this method are also developed by Apostolico and Bejerano [7].

4.3.1 Building an Efficient PST

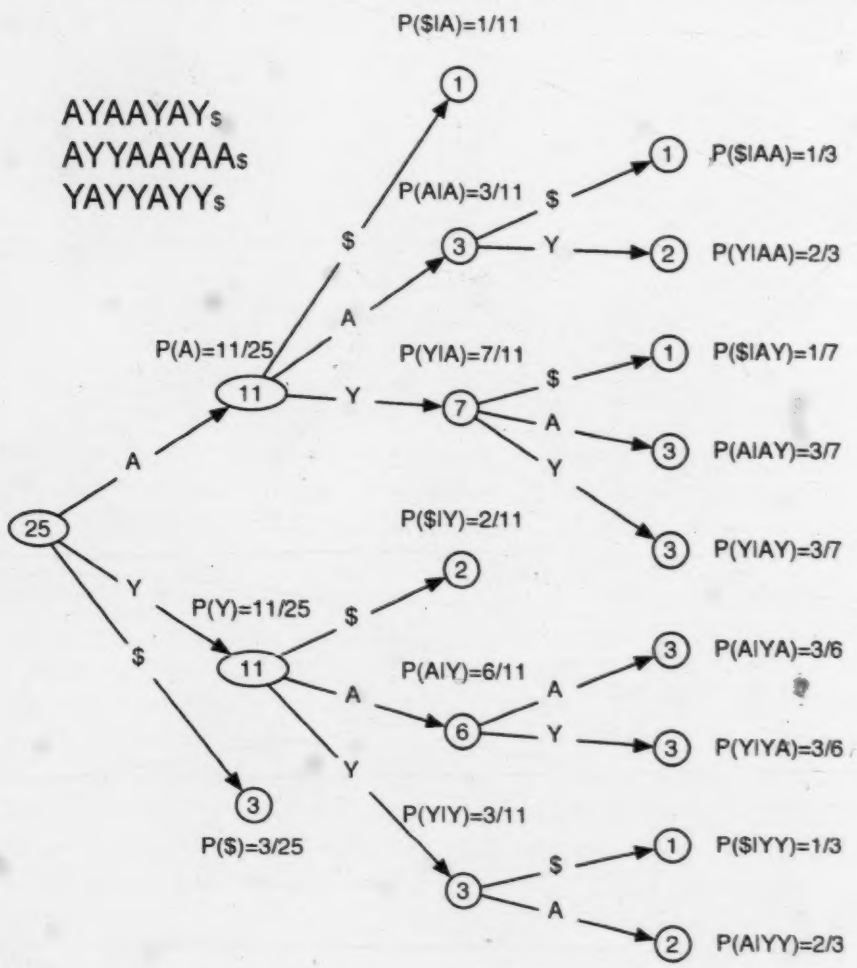
bPSTs have been implemented in order to efficiently look up probabilities for each history in a VMM by storing the histories of the training sequences in the PST. Efficient PSTs (ePSTs) gain efficiency by instead representing the training sequences themselves in the PST. An example ePST is shown in Figure 4.7 which has been constructed from the training sequences 'AYAAYAY', 'AYYAAYAA', and 'YAYYAYY'. To reduce the size and complexity of the example, the tree has been pruned to a maximum history of 2 (see Section 4.3.3 on pruning). The conditional probability estimates are unsmoothed (see Section 4.3.4 on smoothing).

Given knowledge of the algorithms for building a classical suffix tree, the construction of the ePST is a natural extension of that prior work. In order to build the ePST, a generalized suffix tree is built from the training sequences (as in Figure 4.4). Using Ukkonen's algorithm, this tree can be built in linear time and space $O(m)$ with respect to the total length m of the training sequences. In a single depth-first traversal of the tree (also $O(m)$ time) the number of occurrences of each substring represented by each node in the tree can be counted. This corresponds to the number of leaves found below the node in the unpruned suffix tree[4].

[4]Note that since the number of nodes in the tree increase linearly with the total length of the training sequences, we can store single values at each node without affecting the overall linear space or linear time requirements of the algorithm (although the practical time and space requirements may be affected).

Figure 4.7: An Efficient Probabilistic Suffix Tree

The diagram shows an ePST which has been constructed from the training sequences 'AYAAYAY', 'AYYAAYAA', and 'YAYYAYY'. For brevity, the tree shown has been pruned to a maximum history of 2 amino acids. Counts of subsequences which occur in the training sequence are displayed within each node. Note that these counts are not equal to the number of leaves below each node because the tree has been pruned. The counts are equal to the number of leaves below the node in the *complete* suffix tree. Conditional probabilities which result from the counts are also displayed. The suffix links are not shown. The '$' character indicates the end of each string.



Recall from Section 4.2.1 that the conditional probabilities needed for prediction can be calculated from the counts of the subsequences (for convenience, Equation 4.2 is repeated here).

$$\tilde{P}(x_i|x_{i-k}\ldots x_{i-1}) = \frac{C(x_{i-k}\ldots x_{i-1}x_i)}{C(x_{i-k}\ldots x_{i-1})} \tag{4.11}$$

The number of occurrences of the substring $x = x_{i-k}\ldots x_i$ can be obtained by starting from the root of the tree and following the path matching the string (see Figure 4.7). The number of occurrences of the immediate prefix of that string $x_{i-k}\ldots x_{i-1}$ is the parent of the node reached. As such, the conditional probability for each residue given its history can be calculated by dividing by the count at its parent (again see Figure 4.7).

$$C(x_{i-k}\ldots x_i) = count(node(x_{i-k}\ldots x_i)) \tag{4.12}$$

$$node(x_{i-k}\ldots x_{i-1}) = parent(node(x_{i-k}\ldots x_i)) \tag{4.13}$$

$$\tilde{P}(x_i|x_{i-k}\ldots x_{i-1}) = \frac{count(node(x_{i-k}\ldots x_{i-1}x_i))}{count(node(x_{i-k}\ldots x_{i-1}))} \tag{4.14}$$

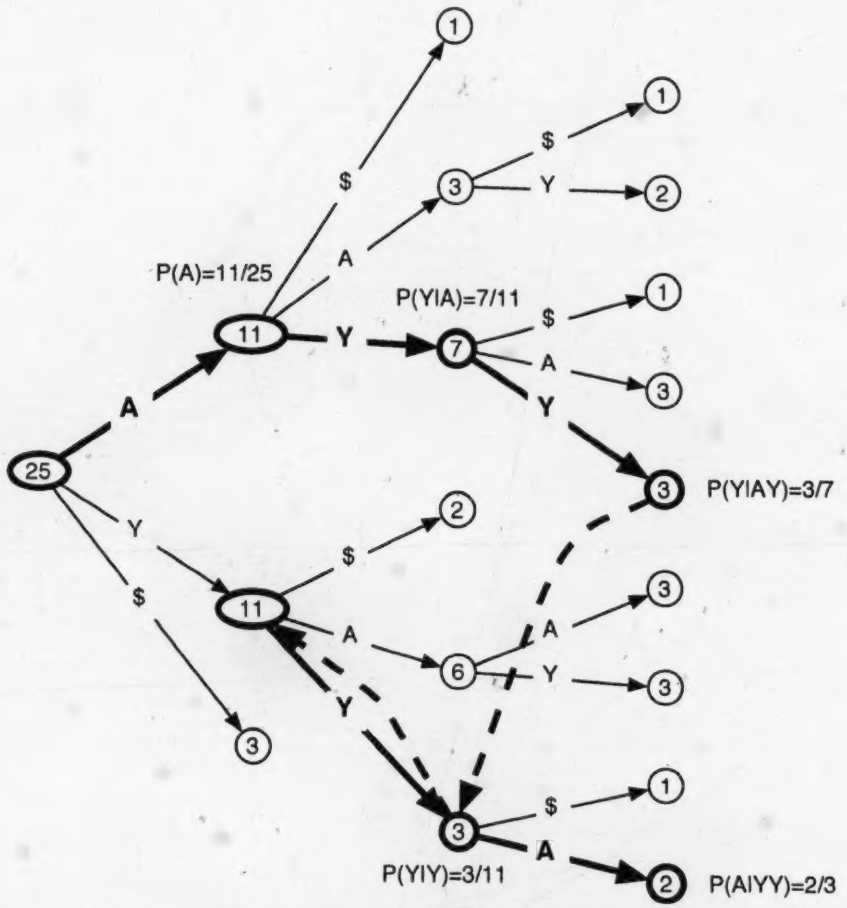
This can be done for the entire suffix tree with another linear time traversal. Each probability can be stored in the node associated with it using an overall linear amount of extra space. Following this traversal, we are left with an efficient PST (see Figure 4.7). For the compressed branches that contain more than one symbol along an arc, the counts for all symbols along the arc are the same. No additional information needs to be stored for these symbols. In a compressed edge the unsmoothed probability of each symbol after the first must be 1.0 because all the symbols in that edge must have the same count (otherwise, the edge would not be compressed).

4.3.2 Prediction with an Efficient PST

Prediction of the probability of a sequence using an efficient PST is possible in linear time with respect to the length of the test sequence. The probabilities are obtained for each position by simply following the path representing the sequence from the root of the ePST (see Figure 4.8). The conditional probabilities are stored at the location reached with each position. When at a node, the conditional probability is equal to the precalculated value. When in an arc, the conditional probability is equal to 1. If the entire sequence is found in the tree, the traversal follows a simple path. If the next symbol is not found in the tree, however, it is still possible that the symbol can be found with a shorter history. The next shortest history of a

Figure 4.8: Prediction with an ePST

The predicted probability of the sequence 'AYYYA' is shown as calculated from the ePST in Figure 4.7. The path followed by the algorithm is shown in bold. The suffix links are shown where necessary for the calculation. Note that when following a suffix link the algorithm does not advance along the sequence.



$$P(AYYYA) = P(A)\ P(YIA)\ P(YIAY)\ P(YIY)\ P(AIYY)$$
$$= 11/25 \cdot 7/11 \cdot 3/7 \cdot 3/11 \cdot 2/3$$

substring is its immediate suffix $suf(\cdot)$. Fortunately, the suffix tree contains suffix links which allow us to move to the next shortest history in constant time. These suffix links can be thought of as backoff or failure links which allow us to move quickly to the next shortest history when the current history fails to satisfy certain criteria.

Figure 4.8 demonstrates how to predict the probability of the sequence 'AYY-YA' given the ePST from Figure 4.7. A path corresponding to the test sequence is followed through the tree. Suffix links are followed where data is not available. In this example, suffix links are followed twice when a 'Y' is not available. The first time occurs as the prediction reaches the maximum history of the tree. A suffix link is followed a second time when the probability for a 'Y' given the history is still not found.

As a side effect of the linear time prediction algorithm, once some part of the sequence history is forgotten (by making a Markov assumption), it is forgotten forever. For example, the conditional probability of the amino acid at position 5 might be calculated using the 2 previous residues $P(x_5|x_3x_4)$ rather than the full history $P(x_5|x_1x_2x_3x_4)$. At this point, we've made the Markov assumption that the conditional probability at position 5 is independent of the first two residues x_1x_2. As a result, the conditional probability of the amino acid at position 6 cannot use any history that is further back than x_3. The maximum history length that the next amino acid x_6 can use is $P(x_6|x_3x_4x_5)$ as x_2 and x_1 have already been 'forgotten' by a Markov assumption. This property is a direct result of the ePST linear time prediction algorithm. This independence assumption may or may not hold in real biological sequences.

4.3.3 Pruning

Pruning is used with both bPSTs and ePSTs to reduce the memory requirements of the model. Pruning is used to eliminate nodes that are either beyond a maximum history length L or are not needed for prediction (as determined by some criteria such as requiring a certain number of training examples or a certain probability threshold).

With bPSTs, pruning is guided by a variety of parameters (as discussed briefly in Section 4.2.2 and more extensively by Apostolico and Bejerano [7]). This pruning occurs in two phases. Pruning by the history length parameter L and the minimum probability parameter P_{min} is done by discarding sequences during the selection of strings that are to be placed in the tree. Pruning according to the ratio criteria r and the minimum conditional probability γ_{min} occurs during the tree construction. With the PST variant suggested by Apostolico and Bejerano [7], the entire tree representing the sequences is constructed in one phase. Each node is then marked if it satisfies the required criteria for remaining in the tree as defined by the pruning

parameters. All the pruning is performed in a subsequent step that trims all edges from the tree that are immediately below the deepest unmarked node along each path.

In the current work, pruning by history length L and the minimum probability P_{min} may also be done by trimming the tree post-construction[5]. Pruning which occurs after the counting of substring occurrences has been finished does not interfere with either the probability calculations or the prediction process. Practical experience with the potentially huge space requirements of representing the entire training set in a suffix tree, however, has prompted an improvement in pruning which allows the linear time algorithm to be used in computers with limited memory. For automatic high-throughput protein function prediction, it is particularly essential that training with large sets be practical.

The current work allows for online pruning of nodes and edges beyond the user-specified history length L. As this pruning takes place on the fly, the complete suffix tree for all the training sequences is never built in memory. The tree which is constructed by this online pruning process is equivalent to a tree which has been obtained by post-construction pruning.

The pruning modification involves a change to the manner in which the suffix tree is extended during Ukkonen's algorithm. It is not necessary to describe Ukkonen's entire algorithm in order to explain the pruning process. It is sufficient to mention that the appropriate paths (nodes or edges) in the suffix tree are extended with the addition of each symbol of the sequence. We are concerned with extensions beyond the maximum history length L. This corresponds to a tree depth limit of $L + 1$. There are only two possible extension cases when the algorithm has reached the limit $L + 1$. The extension will occur at either a node or an edge. For each case the solution is reasonably simple. (The counts at each node are initialized to zero as they are created.)

1. If the extension would have occurred at a node of depth $L + 1$, increment the count of the current node and follow the node's suffix link to the immediate suffix (a node of depth L) and continue the algorithm.

2. If the extension would have occurred along an edge at depth $L + 1$, then increment the count of the last node above the edge and use the skip/count trick (see Section 4.1.2) to move to the immediate suffix of the edge.

After the construction of the tree, the counting procedure is a linear time traversal of the tree in which the number of leaves below each node is stored in that node. When pruning has occurred, the counting procedure does not descend beyond nodes which already have counts due to the modified pruning steps above. The counts at

[5]The current work on ePSTs does not prune according to the parameters r or γ_{min}.

those nodes are considered to be the number of leaves that *would have occurred* below it in the tree. The traversal of the tree uses these nodes as if they had already been counted and continues in a depth-first fashion as in the non-pruned tree.

The online pruning procedure does not change the overall time requirements of the algorithm because the number of suffix links followed in this way is bounded by the overall length of the sequence[6]. This pruning method allows for the selection of L so that the memory requirements of the training algorithm can be reduced. This relatively simple but apparently novel pruning method is a contribution to the use of suffix trees for a variety of applications.

The online pruning is used only for satisfying the memory limit parameter L. Pruning based on other criteria is left until after tree construction and may be performed as mentioned above, by testing each node, marking those that should remain in the tree, and pruning the appropriate nodes in a single step.

Although most pruning is typically done during the training phase of ePST construction, it is also possible to defer some pruning decisions to the testing phase. A pruning decision that is made during testing simply results in the algorithm following a suffix link to the next shortest history. Pruning criteria that can be tested in constant time do not affect the overall time bounds of the training or testing algorithms. In practice, however, it may be beneficial to move these tests to one phase or the other depending on the desired computational properties of the testing and training.

4.3.4 Smoothing

When using Markov chains – especially those of higher-order – the question arises of how to deal with zero or near-zero counts for parameter estimation [50]. Although PSTs automatically prune away all zero counts, small non-zero counts still tend to give poor parameters with maximum likelihood estimation. Although the estimates may be poor, we may not want to prune them away completely. Smoothing provides a method by which the negative effect of small training sets and small counts can be mitigated.

A simple approach to smoothing is to define a minimum probability for any event. This approach, taken by Bejerano and Yona [17], ensures that no conditional probability γ given from a bPST is less than γ_{min}. Their algorithm ensures the minimum probability via a combination of pruning any string which would result in a conditional probability of $\gamma + \alpha$ and smoothing those probabilities which do remain in the tree (according to Equation 4.10). Bejerano and Yona also present a slightly different procedure that smoothes by using pseudocounts that correspond to

[6]Each symbol followed increases our depth in the tree by 1. Each suffix link followed reduces our depth in the tree by 1. The number of suffix links followed by the algorithm cannot exceed the length of the sequence.

the frequencies of amino acids in large protein databases rather than a single value γ_{min}.

A large variety of smoothing methods have been developed for Markov chains [50]. Kermorvant and Dupont [51] argue that the simple smoothing procedures used by Bejerano and Yona can be improved upon by using backoff smoothing [50]. They show results which indicate the advantage of backoff smoothing over the smoothing used in bPSTs. The current work also utilizes backoff smoothing, although with small differences from the work of Kermorvant and Dupont, especially where their suggestions would alter the time complexity of the overall training and prediction algorithms.

Backoff smoothing [50] is used to ensure, as bPST smoothing does, that rare but possible events are assigned an appropriate probability even though they may not occur in the training set. Smoothing removes some of the probability mass which maximum likelihood estimation assigns to events which do occur and moves that probability mass to unseen events. Backoff helps to reassign that probability mass in a more intelligent way by weighting the reassigned mass according to the probabilities of a lower-order conditional probability estimate.

The unsmoothed probabilities are obtained from maximum likelihood estimates. The following is repeated for convenience from Equation 4.2.

$$\tilde{P}(x_i|x_{i-k}\ldots x_{i-1}) = \frac{C(x_{i-k}\ldots x_{i-1}x_i)}{C(x_{i-k}\ldots x_{i-1})} \tag{4.15}$$

Backoff smoothing splits the probability estimates into those cases where the count $C(\cdot)$ is non-zero and those cases where the count is zero. A small probability is taken from the non-zero cases and added to the zero count cases. The probability is based on a pseudocount d which is added to the count for each amino acid.

$$P_s(x_i|x_{i-k}\ldots x_{i-1}) = \begin{cases} \dfrac{C(x_{i-k}\ldots x_i)+d}{C(x_{i-k}\ldots x_{i-1})+d|\Sigma|} & \text{if } C(x_{i-k}\ldots x_i)>0 \\ \alpha(x_{i-k}\ldots x_{i-1}) & \\ \cdot P_s(x_i|x_{i-k+1}\ldots x_{i-1}) & \text{otherwise} \end{cases} \tag{4.16}$$

Since the conditional probabilities for all values of x_i must sum to 1, we can calculate Δ, the total amount discounted by the pseudocount d from the non-zero cases. We need a definition for α. The full derivation is found in Appendix C. The following definition will suffice to show that α is dependent only on the history $x_{i-k}\ldots x_{i-1}$.

$$\alpha(x_{i-k}\ldots x_{i-1}) = \frac{1 - \sum_{a\in\Sigma, C(x_{i-k}\ldots x_{i-1}a)>0} P_s(a|x_{i-k}\ldots x_{i-1})}{1 - \sum_{a\in\Sigma, C(x_{i-k}\ldots x_{i-1}a)>0} P_s(a|x_{i-k+1}\ldots x_{i-1})} \quad (4.17)$$

This smoothed probability has several attractive properties. It is defined recursively and is valid for any history length k. The pseudocount d is basically a prior on the probabilities. The strength of the prior is increased with the size of the pseudocount. In the case shown here, the prior is uniform across all the possible symbols in the alphabet (usually all the amino acids). This could be altered in future work to weight the prior based on the relative frequencies of the amino acids in large protein databases. The smoothing parameters depend only on the next shortest history and can be calculated efficiently. The backoff criteria are flexible and could be changed to backoff under more stringent conditions. The smoothing and backoff method presented here has been implemented in the current work [36]. One of the drawbacks of this type of smoothing is that the calculations for the parameter α cannot be cached in the ePST using linear space. This is because the values in Equation 4.17 may be different for different residues along an edge in the tree. Since we cannot cache different values along the edge and maintain the constant space of an edge in memory, this would require greater than linear space, and thus greater than linear time with respect to the total length of the training sequences. The smoothing values can still be calculated during prediction and this can be done in linear time with respect to the length of the test sequence. The computation required to do this, however, is significant.

4.3.5 Equivalence of PST Implementations

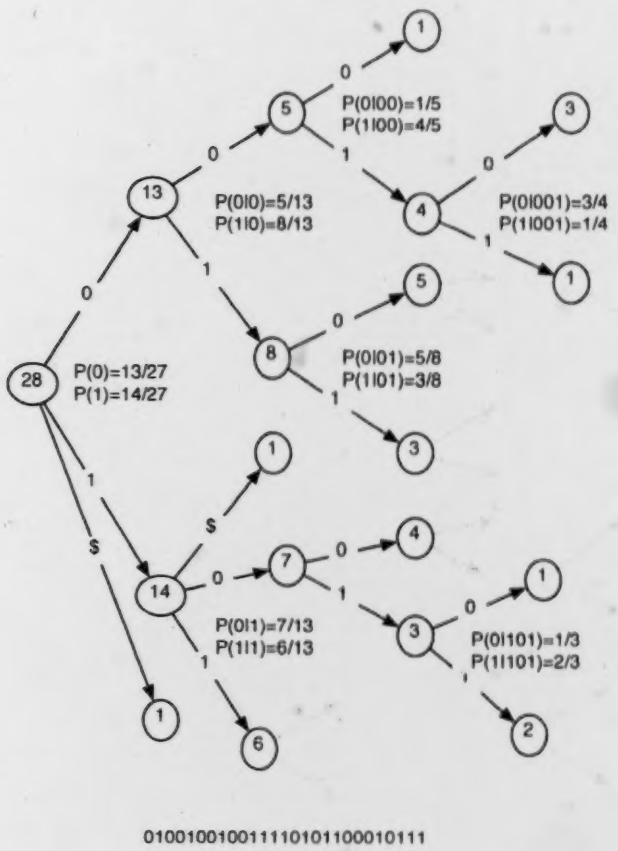
The two implementations of PSTs are able to represent the same information. In essence, following the path of a history in a bPST is equivalent to following the suffix links of an ePST in reverse. Apostolico and Bejerano [7] have also discussed the equivalence of the PST implementations.

Although the following illustration is not rigorous, it will serve to provide some intuition about the equivalence of the PSTs. The pruned ePST in Figure 4.9 may be used to make the same predictions as the bPST in Figure 4.6. We will diagrammatically demonstrate the conversion from an ePST to a bPST. Most of the steps in the conversion involve only the rearrangement of the nodes in the graph, with the exception of a final step which shows where bPSTs lack the additional structural information that ePSTs maintain.

In order to diagrammatically convert a fragment of the ePST to a bPST, we will follow the steps below. Conversion of the entire tree as shown in Figures 4.9 and 4.6 is demonstrated in Figure B.4.

Figure 4.9: Another Efficient Probabilistic Suffix Tree

This ePST represents the same training sequence data as Bejerano's example [15] which is reproduced in Figure 4.6.



1. Show the suffix links (dotted lines) of the ePST. Note the original probabilities from the ePST. We will see them again in the final bPST. The addition of the suffix links to the Figure is done in two steps for clarity. The transition from Figure 4.10-1 to Figure 4.10-2 shows the addition of the first three suffix links. Figure 4.10-3 shows the remaining suffix links.

2. Rearrange the nodes and edges such that the suffix links determine the placement of the nodes in the graph structure. The suffix links become the significant (hard) edges and the other arcs bend as needed. This rearrangement occurs between Figures 4.10-3 and 4.10-4.

3. Flip the tree along the horizontal axis. Figure 4.10-4 to Figure 4.11-5.

4. Calculate the probabilities corresponding to each internal node. Figure 4.11-5 to Figure 4.11-6. These probabilities are the same as those that we noted in the corresponding ePST from the first step.

5. Remove the edges (solid lines) of the ePST and prune extra nodes. Figure 4.11-6 to Figure 4.11-7.

It can be seen that the remaining edges that now make up the bPST are the reverse suffix links of the ePST. bPSTs lack the edges which enable the ePSTs to be used for training and prediction in linear time.

Figure 4.10: Conversion of an ePST to a bPST

This step-wise conversion of an ePST to a bPST is described in the text. Although not a rigorous proof, this example shows how the suffix links (dotted lines) in the ePST are equivalent to (but reversals of) the branches in a bPST.

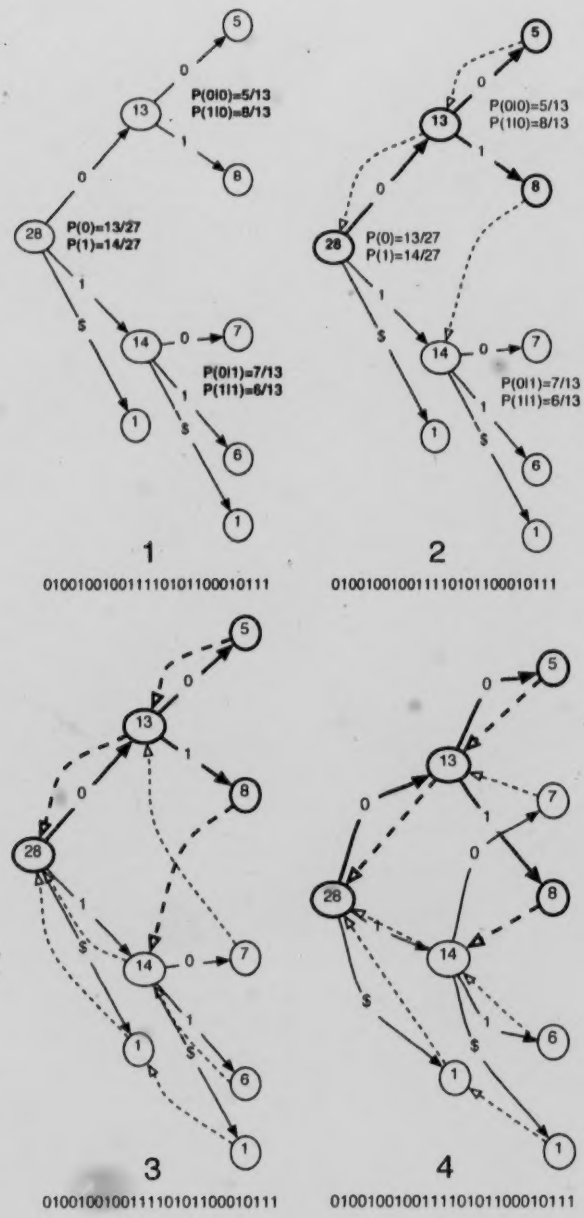
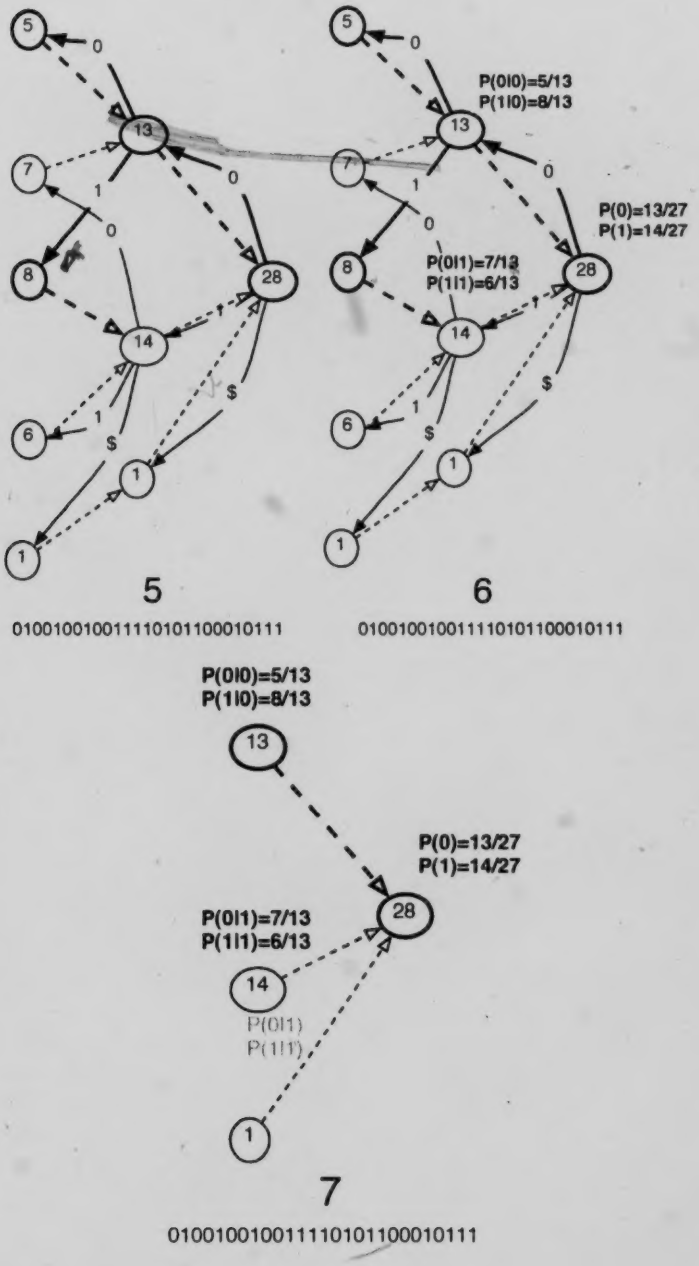


Figure 4.11: Efficient Probabilistic Suffix Tree



Chapter 5

Experiments for Efficient Protein Function Prediction

It has been written that '... the axe is laid unto the root of the trees: every tree therefore which bringeth not forth good fruit is hewn down, and cast into the fire'.[1] The following experiments will evaluate various tools for protein function prediction, including probabilistic suffix trees. The utility of each method will be determined by its classification performance and computational efficiency.

5.1 Datasets

In order to evaluate the classification performance of various methods for high-throughput protein function prediction, each was assessed on a variety of test sets. Each test set is composed of sequences and their associated class labels. The test sets have been chosen from data sources where the confidence of correct labelings is relatively high. Despite this, the datasets contain biological data that has been annotated by a number of different researchers and methods. As such, some noise, inconsistency, and missing information is expected in the data and the labels.

Each of the following datasets was selected in order to compare and contrast the properties of protein function prediction methods on problems of biological interest.

- *pkinase* and *metallothio* Pfam Families — There are many specific protein functions that are of biological interest to many researchers. Two families of the Pfam database [13], one numerous and one with fewer members, were selected as examples of specific protein function prediction. The families are the *pkinase* protein family (PF00069) and the *metallothio* protein family (PF00131). The *pkinase* family is a large group of eukaryotic proteins (Pfam

[1] Luke 3:9

notes more than 10 000 examples) which are protein kinases[2]. The *metallothio* family is a much smaller group (about 170 examples noted by Pfam) of metallothioneins[3]. The two families were chosen in part because of their performance differences with regard to PSTs. As evaluated by Bejerano and Yona [17], the pkinase family was classified poorly with bPSTs and very well with HMMs [17]. In the same set of experiments, the metallothio family was distinguished well by bPSTs but more poorly by HMMs. The families were also combined for some tests to observe performance from *mixed* classes[4]. The three Pfam test sets (*pkinase*, *metallothio*, and the *pkinase/metallothio mixed* class) are all generated from a random selection of about 1/4, or 25912 sequences, of the SwissProt database [19] (version 42, also see below). The positively labeled instances from the training set are those that are designated by Pfam as being in the 'full' set of family sequences (as opposed to the 'seed' set which consists of a small subset of sequences that is representative of the family).

- *K^+ Channels* — The voltage-gated potassium channels test set is composed of 77 proteins selected from 4 mutually exclusive classes of potassium channels. The classes 'Kv1', 'Kv2', 'Kv3', and 'Kv4' each have slightly different functional properties and have been labeled by Warren Gallin [59]. They are typical of sequence sets that a single lab or research group might work with.

- *Gram- Subcell* — The Gram negative bacteria Subcellular location test set is composed of 3960 sequences from SwissProt [19] (version 42) and their associated subcellular locations as annotated in that database. The available subcellular locations include 'cytoplasm', 'extracellular', 'inner membrane', 'outer membrane', and 'periplasm'. Almost every protein has a single subcellular location. There are a few proteins, however, that have multiple subcellular locations — usually when they perform their function at the interface between two subcellular compartments.

- *GO on SwissProt 1/4* — The high-level Gene Ontology [28] data set is made up of a randomly selected group of 25912 sequences from SwissProt ver-

[2]Protein kinases attach phosphate groups to proteins, usually resulting in a change of shape that influences protein function. Protein kinases are often involved in regulating protein activity.

[3]Metallothioneins are small proteins that bind heavy metals. They often consist of high amounts of cysteine residues and low numbers of aromatic amino acids. They are important for metal ion transport.

[4]Mixed protein classes containing distinct protein types may arise for a variety of reasons. The class may be a high-level class that contains several subclasses. The sequences might be contaminated with another class of proteins that shares some properties but are functionally different. Annotation errors and noisy data may also result in mixed classes. In any case, the robustness of a classification method depends on the ability to function under noisy conditions.

sion 42 (about 110 000 sequences). The Gene Ontology (GO) class labels have been taken from EBI's Gene Annotation [27][5]. The Gene Ontology classes used in this data set have been selected from high-level categories of the GO Molecular Function hierarchy (see Figure 1.4). The classes (and their associated GO numbers) are 'binding (0005488),' 'catalytic activity (0003824),' 'hydrolase activity (0016787),' 'lyase activity (0016829),' 'metal ion binding (0046872),' 'nucleic acid binding (0003676),' 'nucleotide binding (0000166),' 'oxidoreductase activity (0016491),' 'signal transducer activity (0004871),' 'structural molecule activity (0005198),' 'transferase activity (0016740),' and 'transporter activity (0005215).' These classes all have many examples in the SwissProt database. Some classes are subclasses of others (see Figure 1.4). Each protein may be a member of one or more GO classes.

These data sets come with an important caveat. In addition to empirical evidence, much of the training data has been acquired and labeled based on sequence analysis using BLAST and HMMs, two of the techniques that we are evaluating. For example, the annotation of new proteins for SwissProt is made much easier when high-scoring BLAST alignments and Pfam HMMs are available. This gives BLAST and HMMer an inherent advantage in the analysis and should be remembered when comparing the results of the classification techniques.

5.2 Evaluation

For each data set the classes are treated as independent classification tasks. When a data set has more than one class, the results are reported for each class individually and then as an overall result for all classes (see Section 2.3.4) using 5-fold cross-validation results (see Section 2.3.2). Due to space limitations, detailed results for the individual classes of each data set have often been placed in Appendix A as referenced.

Accuracy can be a very misleading measure of performance, especially where the class distributions are very imbalanced (where some classes are large while others are very small). Consider a protein family with few members in a very large data set. If that family has 100 positive matches in a data set of 1000, a classifier which predicts that a single true member of the class is positive and that one of the proteins which is not in the class is positive (and all other 998 proteins are negative) will yield an accuracy of 90% (1 true positive, 899 true negatives, 1 false positive, 99 false negatives). The precision would be 50% and the recall would be 1% – obviously poor performance. Consider a second classifier which correctly predicts that all the 100 true members of the family are positive but incorrectly predicts an

[5]Downloaded January 2004.

Table 5.1: Majority Classification

Protein Class	Performance		
pkinase Pfam	Precision —	Recall 0.000	Accuracy 0.985
metallothio Pfam	Precision —	Recall 0.000	Accuracy 0.998
mixed Pfams	Precision —	Recall 0.000	Accuracy 0.984
K^+ Channels (Table A.1)	Precision —	Recall 0.000	Accuracy 0.750
Gram- Subcell (Table A.2)	Precision 0.628	Recall 0.620	Accuracy 0.848
GO on SwissProt 1/4 (Table A.3)	Precision 0.594	Recall 0.253	Accuracy 0.820

additional 100 as positive. The precision would be 50% and the recall would be 100% – apparently a better classifier than the first. The accuracy, however, would again be 90%. When accuracy is used for an imbalanced data set (as often occurs in many protein function prediction tasks – including those that we will consider here) the precision and recall are often crucial measures of performance. Precision, recall, coverage, and accuracy will all be reported here.

The overall performance measures may also seem misleading for the multiple class data sets (*K^+ Channels*, *Gram- Subcell*, and *GO on SwissProt 1/4*) since the issues arising from imbalanced class distributions are multiplied in the overall accuracy. Again, readers may wish to focus more on precision and recall than accuracy. In addition, the complete results for each class are included in Appendix A and may be referred to for detailed outcomes where necessary.

For each classification task it is important to compare to a baseline classifier. A classifier that randomly predicts positive or negative for each class with an equal probability for each (0.5) will be expected to obtain 50% accuracy on average. We certainly expect our classifiers to exceed this threshold. As each protein function class is typically much less than 50% of the data set, we recognize that a classifier that simply predicts the label of the majority of the data set will achieve better than 50% accuracy. We will use this 'majority classifier' as a simple baseline for the results our data sets. The performance of majority classification for each data set

can be seen in Table 5.1 and is far above the 50% accuracy that would be obtained by a random classifier.

The evaluation of computational efficiency was done using simple randomly generated data sets. The runs were times on a single processor of a two-processor 2.0 GHz Apple Power Mac G5 with 3.5 GB of RAM running Mac OS X Version 10.3.3. All times are reported in seconds.

5.3 Alignment-Based Methods

5.3.1 BLAST

As BLAST nearest-neighbor classification is the most commonly used classification scheme (especially for biologists using non-automated sequence analysis), we consider the results of using BLAST over the given datasets. To evaluate BLAST as a classifier, the test set is aligned against the training set. For each test sequence we predict the class label to be the label of the sequence with the best scoring (smallest E-value) alignment match in the training set. When no alignment can be made with an E-value of less than 10, no prediction is made. The results for using BLAST as the distance measure for a 1-nearest-neighbor classifier are found in Table 5.2. Experiments with 3-nearest-neighbor classification (data not shown) gave almost identical results.

BLAST is able to predict the correct classification for the vast majority of the proteins in these training sets. In some cases there were no predictions given for a significant number of proteins because no BLAST hits were found below the E-value threshold. Given the very permissive E-value threshold, we expect a relatively high number of spurious alignments (unrelated sequences which score well by chance). A high E-value threshold also gives better coverage. By changing the E-value threshold we can hope to increase precision at the expense of decreased coverage. For example, Figure 5.1 displays how the coverage of the *GO with SwissProt 1/4* data set responds to various E-value thresholds. This cross-validation coverage shows that there are a considerable number of proteins in the SwissProt database that have very similar sequences within the data set. About 20% of our data set aligns to other sequences with E-value scores smaller than 10^{-170} (almost identical). Although this plot tells us how coverage responds to E-value, we would really like to know what E-value to select for optimal classifier performance.

Figures 5.2 and 5.3 show how coverage, precision, recall, and accuracy respond to changes in E-value for the *Gram- Subcell* and *GO with SwissProt 1/4* data sets. Precision is affected very little for E-value thresholds smaller than 10^{-10}. The 'local precision' (the precision over all sequences finding a neighbor within the immediate range – 10^{-n-5} to 10^{-n}) drops most precipitously for E-values greater than 10^{-5}.

Table 5.2: BLAST Nearest-Neighbor

The following data are obtained by 5-fold cross-validation over the given training sets. Alignments with an E-value of greater than 10 were counted as 'no predictions'.

Protein Class	Performance			
pkinase Pfam	Precision 0.848	Recall 0.989	Accuracy 0.990	Coverage 0.992
metallothio Pfam	Precision 0.950	Recall 0.974	Accuracy 0.992	Coverage 0.992
mixed Pfams	Precision 0.856	Recall 0.988	Accuracy 0.990	Coverage 0.992
K^+ Channels (Table A.4)	Precision 1.000	Recall 1.000	Accuracy 1.000	Coverage 1.000
Gram- Subcell (Table A.5)	Precision 0.931	Recall 0.900	Accuracy 0.964	Coverage 0.996
GO on SwissProt 1/4 (Table A.6)	Precision 0.934	Recall 0.934	Accuracy 0.968	Coverage 0.992

Figure 5.1: BLAST Coverage for 5-fold Cross-Validation, *SwissProt 1/4*

The following graph shows the coverage achieved with 5-fold cross-validation with test sets against training sets for BLAST scores.

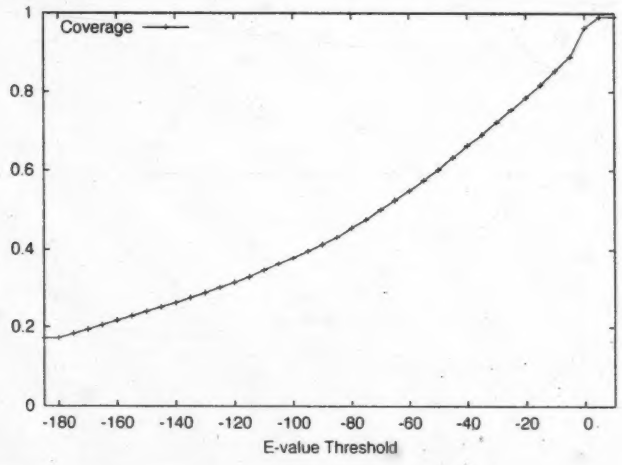


Figure 5.2: BLAST Nearest-Neighbor Performance Response to Varying E-value Threshold, *Gram- Subcell*

The response of performance measures to changing E-value thresholds is shown in the plot. As the E-value is made less stringent (larger) the recall increases and the precision decreases. The local precision at an E-value of 10^n is the precision of those classifications which are made when a nearest neighbor is found within the immediate range of measurement (10^{-n-5} to 10^{-n}, marked by tics on the scale).

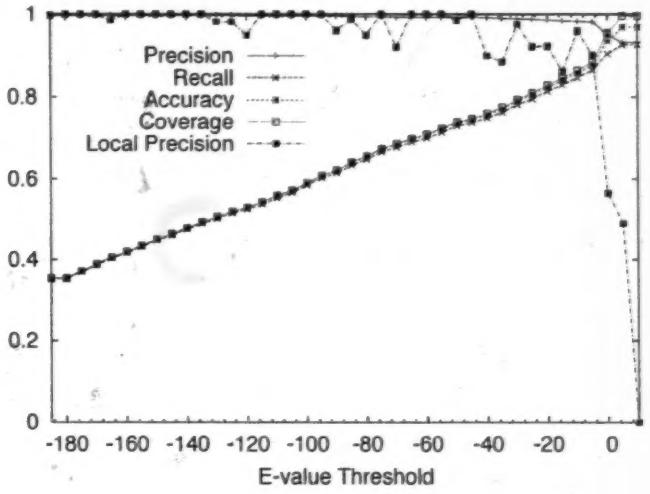


Figure 5.3: BLAST Nearest-Neighbor Performance Response to Varying E-value Threshold, *GO with SwissProt 1/4*

The response of performance measures to changing E-value thresholds is shown in the plot. As the E-value is made less stringent (larger) the recall increases and the precision decreases. The local precision at an E-value of 10^n is the precision of those classifications which are made when a nearest neighbor is found within the immediate range of measurement (10^{-n-5} to 10^{-n}, marked by tics on the scale).

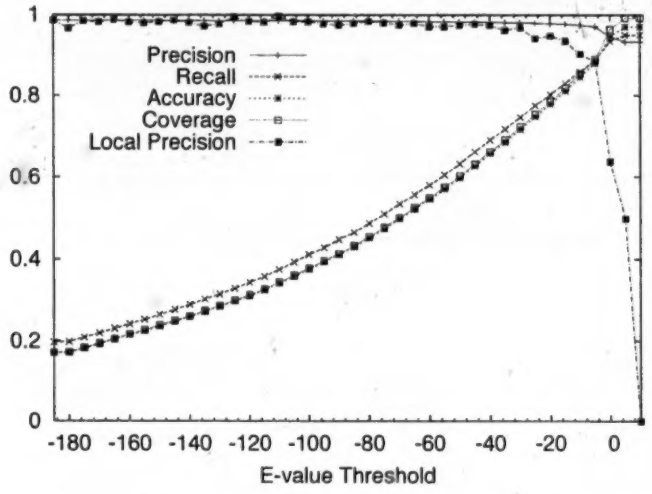
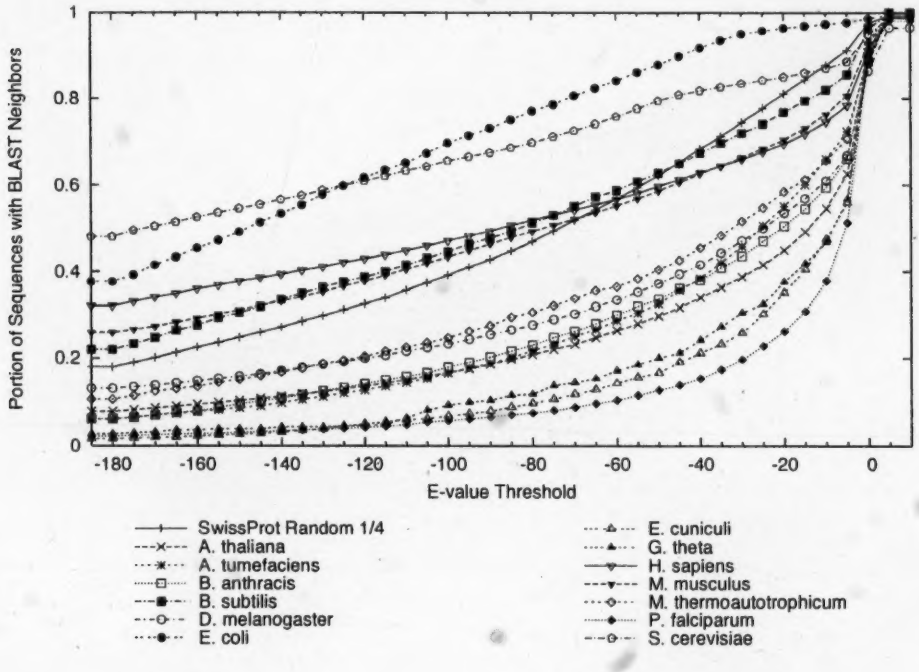


Figure 5.4: BLAST Coverage of SwissProt for Various Organisms

The chart displays the number of sequences which can be aligned to a sequence in SwissProt with a score better than the given threshold. Some organisms are well represented in SwissProt and obtain many exact matches (E-value of 10^{-180} or better). The number of similar sequences which occur in SwissProt is much higher for well-studied organisms.



The coverage is greatly reduced over the entire range of decreasing E-values. For these classification problems, performance may be optimized by choosing an E-value threshold somewhere in the range of 10^{-10} to 10^{0}, depending on the relative importances of coverage and precision for the task at hand.

These cross-validation results indicate that BLAST will have excellent performance for predicting subcellular location and Gene Ontology categories. It is important to remember the caveat mentioned above – many label annotations have been made using BLAST alignment information. Thus, many of the sequences in SwissProt have similar sequences which are already in the database (as we observe in Figure 5.4) and the BLAST results may in some cases only confirm annotations that were done using BLAST. We hope that as SwissProt is manually curated that the effects of this will be small.

If we believe that the precision of a BLAST nearest-neighbor classification will hold according to the results observed in Figures 5.2 and 5.3, we will feel rela-

tively confident about predictions based on neighbors that have an alignment with an E-value better than 10^{-5}. To develop an intuition about how well using BLAST nearest-neighbor with SwissProt as a labeled training set will work in general, we can examine the coverage of the method on various protein sets for entire organisms (proteomes).

Figure 5.4 shows how coverage responds to E-value threshold over proteomes from a variety of completely sequenced organisms. Each proteome was aligned to the SwissProt database by BLAST and the number of sequences with at least one BLAST hit better than each E-value threshold are recorded. The BLAST coverage of our *Random 1/4* of SwissProt against SwissProt is also plotted. We observe stark contrasts between organisms. This figure shows that the coverage of commonly used model organisms such as *E. coli* and *S. cerevisiae* is very high in SwissProt. It is very likely, however, that researchers will desire to predict protein function on proteomes that have been less studied, such as *P. falciparum* and *E. cuniculi*. These organisms have comparatively low coverage for an E-value of 10^{-5}. As such, BLAST nearest-neighbor prediction is much less likely to work well for these relatively unstudied organisms. Because of the severity of the coverage problem, it is important to study methods that can work around this issue. One way we can do this is to try to increase the robustness of BLAST nearest-neighbor when coverage is reduced (Proteome Analyst uses some techniques which accomplish this). We will also examine methods which do not rely on sequence alignment for the prediction. There are a variety of sequence patterns such as HMMs and PSTs which offer alignment-free predictions.

5.3.2 Proteome Analyst

Proteome Analyst (PA) [90, 91, 63, 62] is a variation of the BLAST nearest-neighbor approach which strives to use additional information to increase precision and coverage. PA aligns the training sequences against the sequences in the SwissProt database and selects a small group of the nearest neighbors (usually three). The annotations of these neighbors are then extracted from the SwissProt database, resulting in a group of keywords that are related to those sequences. These keywords are typically related to protein function and properties. These keywords become a vector of features that can be used by a classifier such as naïve Bayes (see Section 2.4.3). Once a classifier has been created using the keywords from the training set, the process of obtaining BLAST alignments against SwissProt is repeated for the test sequences. The SwissProt keywords are again extracted, creating a feature vector for each test protein. A naïve Bayes classifier which has been built using the features of the training set is then used to predict the class of the each test protein given its feature vector. The results of using Proteome Analyst on the data sets are seen in Table 5.3. Since the proteins in many of the data sets are drawn from Swis-

Table 5.3: Proteome Analyst

The following data are obtained by 5-fold cross-validation over the given training sets.

Protein Class	Performance			
	Precision	Recall	Accuracy	Coverage
pkinase Pfam	0.937	0.903	0.997	0.999
K^+ Channels (Table A.7)	1.000	1.000	1.000	1.000
Gram- Subcell (Table A.8)	0.960	0.978	0.987	1.000
GO on SwissProt 1/4 (Table A.9)	0.919	0.956	0.970	0.994

sProt, PA simulates the case where these proteins are not in SwissProt by ignoring the best BLAST hit (which is assumed to be the sequence itself).

The use of the SwissProt annotations allows Proteome Analyst to use a slightly more permissive E-value threshold (10^{-3} vs. 10^{-5}) than a higher-precision BLAST search while still obtaining very precise results. This increases the coverage of the method without sacrificing precision. Alternately, the same E-value threshold may be used (as with BLAST nearest-neighbor) to increase the precision without sacrificing recall. In addition to increased classification performance, PA also allows a very intuitive explanation of each protein's classification through the use of keywords from the SwissProt annotations. This explanation facility does not improve the performance of the classification, but it does increase the transparency for biologists who would like to understand how each protein was classified.

Although the Proteome Analyst approach improves results over a BLAST nearest-neighbor classifier, it still does not address the more difficult coverage issue of dealing with proteomes that have very few alignment matches to the SwissProt database. For this, we turn our focus to pattern-based methods.

5.4 Pattern-based methods

5.4.1 HMMer

HMMs (Section 3.3.2) provide probabilistic modeling of protein families. They have been shown to be very useful for relatively small and well-characterized protein families, as demonstrated by the popularity of the Pfam database. The training

Table 5.4: HMMer

The following data are obtained by 5-fold cross-validation over the given training sets.

Protein Class	Performance			
pkinase Pfam	Precision 0.992	Recall 0.992	Accuracy 1.000	Coverage 1.000
metallothio Pfam	Precision 1.000	Recall 1.000	Accuracy 1.000	Coverage 1.000
mixed Pfams	Precision 0.906	Recall 0.906	Accuracy 0.997	Coverage 1.000
K^+ Channels (Table A.10)	Precision 1.000	Recall 1.000	Accuracy 1.000	Coverage 1.000
Gram- Subcell (Table A.11)	Precision 0.592	Recall 0.592	Accuracy 0.835	Coverage 1.000
GO with SwissProt 1/4	Did not finish.			

sequences in each training set were multiply aligned using ClustalW [92]. In order to simulate high-throughput and automatic protein function prediction, no further refinements of the multiple sequence alignment was made[6]. The multiple sequence alignment is then passed to HMMer [35] software (hmmbuild) which creates an HMM corresponding to the alignment. The HMM is then matched (using hmmpfam) against each test protein and a score is assigned. The test proteins are then evaluated against the known labels.

Since no score threshold is defined beforehand for determining which instances are positive or negative, the threshold is determined by the *isopoint* criteria [17]. The isopoint is determined by sorting the proteins according to score. The proteins which have HMM scores that are better than a certain threshold are considered matches. The performance of the classifier is evaluated at each possible threshold (as in an ROC curve, Section 2.3.4). The threshold at which the number of false positives is equal to the number of false negatives is the *isopoint*.

The results of HMM classification for the the various data sets are seen in Table 5.4. The performance measures for each are reported at the isopoint.

HMMs perform extremely well on the Pfam families. This is to be expected, as Pfam families are defined by HMMer hidden Markov models. HMMs are particularly well suited to protein families for which a good multiple alignment can be built. Notably, the performance of HMMs exceeds that of the alignment-based

[6]Alignments used in the Pfam database may be refined and optimized for performance.

Table 5.5: Pfam and PROSITE with Naïve Bayes

The following data are obtained by 5-fold cross-validation over the given training sets.

Protein Class	Performance			
	Precision	Recall	Accuracy	Coverage
K^+ Channels (Table A.12)	0.987	1.000	0.997	1.000
Gram- Subcell (Table A.13)	0.899	0.793	0.913	0.972
GO with SwissProt 1/4 (Table A.14)	0.919	0.878	0.928	0.961

methods for these protein classes.

The mixed Pfam families demonstrate how profile HMMs deal with mixed protein classes. In essence, the smaller *metallothio* family is not modeled in the HMM since a multiple sequence alignment which represents both the *pkinase* and *metallothio* families well cannot be made. The problem of modeling multiple families is exacerbated in the *Gram- Subcell* and *GO with SwissProt 1/4* data sets. It is apparent that HMMs are not well suited to modeling larger and more divergent families such as those in the *Gram- Subcell* and *GO with SwissProt 1/4* training sets. It is unclear what a multiple sequence alignment or the corresponding HMM represents in these cases. In addition to poor classification performance on these sets, the computational requirements of the multiple sequence alignment (using ClustalW) are excessive for very large protein classes (see Section 5.5 for more discussion of this topic). For the largest classification problem, *GO with SwissProt 1/4*, the alignment process was prohibitive[7]. Despite these limitations, we can use HMMs in alternative way. In the following results, groups of HMMs representing smaller protein classes are used in combination to predict larger, more general protein classes.

5.4.2 Pfam and PROSITE Pattern Databases

Although finding HMMs and regular patterns for very large families of sequences can be prohibitively expensive for high-throughput classification tasks (see Sections 5.4.1 and 3.2.4), databases of previously identified patterns can be used to avoid this expense. Using databases of this type allows us to take advantage of large amounts of prior knowledge in protein classification. The Pfam [13] and PROSITE

[7]ClustalW ran for longer than two days using more than a gigabyte of memory without finishing the alignment. The run was eventually terminated before completion.

Table 5.6: Pfam and PROSITE with SVM

The following data are obtained by 5-fold cross-validation over the given training sets.

Protein Class	Performance			
	Precision	Recall	Accuracy	Coverage
K^+ Channels (Table A.15)	0.987	0.974	0.990	1.000
Gram- Subcell (Table A.16)	0.972	0.865	0.946	0.972
GO with SwissProt 1/4 (Table A.17)	0.963	0.907	0.942	0.961

[84] databases contain patterns for a large variety of protein families. Although these protein families have been designed for classification tasks that may be quite different from a user-defined classification task, they can still be very useful for classification. It is possible to use the patterns in a way which is more robust than simple rule-based approaches such as InterProScan [8] by incorporating well known machine-learning approaches (see Section 2.4).

Each sequence in the data set is run against the Pfam and PROSITE pattern databases to find all patterns that match the protein. The matching patterns form a feature vector for that instance. The feature vectors of the training set can then be used by a machine learning technique to create a classifier. This classifier is used to predict the class of a test sequence based on its feature vector of matched patterns. Because the processing time required to run Pfam and PROSITE against the larger *Gram- Subcell* and *GO with SwissProt 1/4* data sets is extensive and each sequence from these sets is found in the SwissProt database, the matching patterns for these sets were extracted from the SwissProt annotations for each sequence. The resultant data is equivalent to the procedure described above, but is much less computationally expensive.

The results of using the Pfam and PROSITE database are seen for the data sets using a naïve Bayes classifier in Table 5.5. The same set of features can be used for other machine learning techniques such as the support vector machines (SVMs). The results seen in Table 5.6 were obtained using the SMO implementation of SVMs in the WEKA machine learning suite [98]. The Pfam data sets were not run using this specific method as it is apparent from the previous results that the individual HMMs for the respective families perform very well for these sets.

The use of pattern databases and machine-learning classifiers returns promising results on the data sets. As with alignment-based techniques, some coverage prob-

lems arise where some of the sequences match no patterns. Where this happens, the classification is conservative and makes no prediction. Unfortunately, we are left relatively powerless to increase the coverage for a particular dataset except by discovering and contributing new protein families to the databases. Despite the fact that the coverage of the Pfam database has increased significantly in recent years and continues to increase, it is unlikely that the coverage will be complete for all new families of sequences which we may want to classify. In order to deal with the low coverage of these pattern databases and to tune the families for specific tasks, we would still like to use a technology which will allow us to quickly build a pattern that represents an entire protein family. We have seen the limitations of HMMs in that regard. PSTs (see Section 4.2) are possible models that may facilitate high-throughput and automatic prediction.

5.4.3 Probabilistic Suffix Trees

PSTs (Section 4.2) are probabilistic models that can be used for a variety of sequence prediction tasks. Like HMMs, they can be built to represent relatively well conserved sequence families [17]. In contrast to HMMs, however, they do not require a multiple sequence alignment.

bPSTs

Bejerano [15] has recently released an implementation of PSTs (here called 'bPSTs'). This implementation allows the use of PSTs for protein function prediction. A bPST is built from the positive instances of the training set. Given a test sequence, the bPST calculates the probability of that sequence for both the PST and a null (random) model. The result is a log-odds score for each sequence. To eliminate any dependence on the length of the sequences, the log-odds scores are normalized by dividing by the sequence length. The predictions for each sequence are then sorted and evaluated at the log-odds score which defines the *isopoint* (as described above for HMMs). Table 5.7 shows the results of using Bejerano's PST program over the data sets. The parameters used are exactly those which were previously published for Pfam prediction [17] and confirmed by Bejerano to be the most reasonable parameter settings [14].

The results using bPSTs are mixed. As found by Bejerano and Yona [17], bPSTs perform well for the *metallothio* family (approaching that of HMMs) but not for the *pkinase* family. The results for the *mixed* family is essentially a weighted average of the two families and the poor performance mainly echoes the results for the larger *pkinase* family. The results for *Gram- Subcell* are encouraging, exceeding both HMMs and the classifiers that use the Pfam and PROSITE databases. The bPSTs had not only better classification performance but better time performance

Table 5.7: bPST

The following data are obtained by 5-fold cross-validation over the given training sets.

Protein Class	Performance			
pkinase Pfam	Precision 0.517	Recall 0.517	Accuracy 0.986	Coverage 1.000
metallothio Pfam	Precision 0.974	Recall 0.974	Accuracy 1.000	Coverage 1.000
mixed Pfams	Precision 0.553	Recall 0.553	Accuracy 0.986	Coverage 1.000
K^+ Channels (Table A.18)	Precision 1.000	Recall 1.000	Accuracy 1.000	Coverage 1.000
Gram- Subcell (Table A.19)	Precision 0.796	Recall 0.796	Accuracy 0.917	Coverage 1.000
GO with SwissProt 1/4 (Table A.20)	Precision 0.555	Recall 0.555	Accuracy 0.826	Coverage 1.000

as well. The time required for bPSTs on this data set was much less than for the other techniques (see Section 5.5 for related results). The *GO with SwissProt 1/4* results, however, are poorer than classifiers using the Pfam and PROSITE databases and barely above the performance of a random classification. bPSTs are better than HMMs for this task, if only because they were able to finish the classification. Alignment-based methods still far exceed the classification performance of bPSTs, but only when close alignments can be found.

bPSTs show some promise for high-throughput function prediction. Classification performance is excellent in some cases. They are able to represent a large set of sequences as a single probabilistic pattern where HMMs cannot. The time requirements are much less than for the competing pattern and alignment-based methods. bPSTs still exhibit weaknesses, however. Although some examples of classification performance are good, others remain poor. This may be the result of poorly tuned parameters, but a search of the parameter space for each protein family would likely negate any benefits of improved time performance.

ePSTs

Although bPSTs have shown some encouraging results for protein function prediction, we would like to improve both the classification and time performance of PSTs.

Table 5.8: ePST

The following data are obtained by 5-fold cross-validation over the given training sets. The prediction method is global and evaluated at the isopoint.

Protein Class	Performance			
pkinase Pfam	Precision 0.594	Recall 0.594	Accuracy 0.988	Coverage 1.000
metallothio Pfam	Precision 0.897	Recall 0.897	Accuracy 1.000	Coverage 1.000
mixed Pfams	Precision 0.649	Recall 0.649	Accuracy 0.989	Coverage 1.000
K^+ Channels (Table A.21)	Precision 1.000	Recall 1.000	Accuracy 1.000	Coverage 1.000
Gram- Subcell (Table A.22)	Precision 0.849	Recall 0.849	Accuracy 0.939	Coverage 1.000
GO with SwissProt 1/4 (Table A.23)	Precision 0.765	Recall 0.765	Accuracy 0.908	Coverage 1.000

Recent publications suggest that the classification performance can be improved by better smoothing [51] and better scoring (through the use of local predictions [89]). Although Bejerano and Yona [17] suggest and test some cases of local prediction with promising results, it is not implemented directly in Bejerano's program. The implementation of local prediction implemented by Sun and Deogun is not publicly available. Kermorvant and Dupont's improved smoothing techniques have not been implemented in a PST.

There have also been suggestions for a great improvement in the theoretical computational requirements of the PST algorithm. Although Apostolico and Bejerano [7] present linear time methods of training and prediction, they make no program available for doing this. Sun and Deogun [89] mention an implementation of this technique, but do not make it available. For an implementation of these ideas, we turn to the efficient PST (ePST) implementation developed in the course of the current work.

The current implementation of ePSTs (see Section 4.3) implements smoothing similar to that which has been used in Markov chains for natural language processing [50] and which has also been suggested by Kermorvant and Dupont [51]. Pruning is less aggressive than that implemented by bPSTs and allows all parameters that do not exceed the maximum history length L and have at least one occurrence

in the training set to remain in the tree. Simple local prediction was tested using cumulative log-odds scores over various window sizes. The more complex local predictions of Sun and Deogun were not tested.

Training an ePST proceeds as for HMMs and bPSTs – by building an ePST from the training sequences. Predictions result in a ePST score for each test sequence which is compared to a random model (a 0-th order Markov model based on the negative training data) for each sequence position. To obtain global scores, the scores at each position are added to give a cumulative log-odds score over the entire sequence, which is then divided by the sequence length to reduce sequence-length effects (as for bPSTs). For a local prediction at a specific amino acid, the log-odds scores are cumulated over a fixed length window of preceding residues. This local prediction can be done for each position over the entire sequence in linear time and the maximum local prediction score becomes the score assigned to the entire sequence. For evaluation, each sequence is sorted by score and performance is measured at the *isopoint* (as for HMMs and bPSTs above).

The results of global prediction using ePSTs are seen in Table 5.8. ePSTs outperform bPSTs in every case except for the *metallothio* proteins. Although ePSTs are equivalent to bPSTs in general, the implementations of pruning and smoothing differ between the two programs tested here. The less aggressive pruning and more advanced smoothing of the ePST implementation may be responsible for the performance differences. It is also likely that parameter selection could be tuned to obtain better results for each of the algorithms. Due to the time requirements of running bPSTs, an exhaustive parameter search was not done. The ePST implementation performs well enough to approach that of pattern databases in combination with the naïve Bayes and SVM classifiers (and exceed it in the case of the *Gram- Subcell* data set). ePSTs still cannot meet the classification performance of the alignment-based methods but are able to achieve 100% coverage. ePSTs also require much less computational time than alignments.

ePSTs were also examined using local predictions. The results when using a window of 40 amino acids can be seen in Table 5.9. Local prediction improved classification performance in some cases (such as *GO with SwissProt 1/4* and decreased it in others (such as *Gram- Subcell*). For prediction on the more conserved Pfam families the selection of the right window length can improve performance. The change from a window length of 40 to a window length of 200 makes a significant difference. It is difficult, however, to select a window length that will improve classification over all the classes involved in the more general classification schemes.

Comparisons that demonstrate the performance at the *isopoint* do not necessarily show classification performance in all circumstances. Many researchers may prefer more precision at the expense of recall. Alignment-based methods such as BLAST and Proteome Analyst tend to be more conservative predictors in this way.

Table 5.9: ePST Local Predictions

The following data are obtained by 5-fold cross-validation over the given training sets. A length of 40 amino acids was used for the scoring window in each case except where noted otherwise.

Protein Class	Performance			
pkinase Pfam	Precision 0.674	Recall 0.674	Accuracy 0.991	Coverage 1.000
pkinase Pfam (window length 200)	Precision 0.745	Recall 0.745	Accuracy 0.993	Coverage 1.000
metallothio Pfam	Precision 0.897	Recall 0.897	Accuracy 1.000	Coverage 1.000
mixed Pfams	Precision 0.726	Recall 0.726	Accuracy 0.991	Coverage 1.000
K^+ Channels (Table A.24)	Precision 0.961	Recall 0.961	Accuracy 0.981	Coverage 1.000
Gram- Subcell (Table A.25)	Precision 0.821	Recall 0.821	Accuracy 0.927	Coverage 1.000
GO with SwissProt 1/4 (Table A.26)	Precision 0.780	Recall 0.780	Accuracy 0.914	Coverage 1.000

Table 5.10: ePSTs Evaluated at Maximum F-measure

The following data are obtained by 5-fold cross-validation over the given training sets. The evaluation is taken at the point that maximized the F-measure with the parameter $\beta = 1$.

Protein Class	Performance			
pkinase Pfam	Precision 0.667	Recall 0.772	Accuracy 0.991	Coverage 1.000
metallothio Pfam	Precision 0.907	Recall 1.000	Accuracy 1.000	Coverage 1.000
mixed Pfams	Precision 0.691	Recall 0.810	Accuracy 0.991	Coverage 1.000
K^+ Channels (Table A.27)	Precision 0.963	Recall 1.000	Accuracy 0.990	Coverage 1.000
Gram- Subcell (Table A.28)	Precision 0.937	Recall 0.797	Accuracy 0.948	Coverage 1.000
GO with SwissProt 1/4 (Table A.29)	Precision 0.848	Recall 0.753	Accuracy 0.925	Coverage 1.000

In order to view the tradeoff between precision and recall, ePSTs were also evaluated with a threshold that maximizes the F-measure (see Section 2.3.4, Equation 2.16) having an β parameter of 1[8]. These results can be seen in Table 5.10. The F-measure tends to select a more conservative threshold on the larger datasets, which results in better performance for *Gram- Subcell* and *GO with SwissProt 1/4*. With more conservative thresholds, ePSTs have classification performance that rivals the use of pattern databases in combination with machine learning classifiers. A less conservative threshold is chosen for the smaller datasets and gives no significant improvement in results.

Smoothing affects classification performance. Table 5.11[9] shows the results of using a pseudocount parameter of 0. This parameter setting effectively eliminates smoothing and the resulting classification performance is very poor.

Pruning affects both speed and classification performance. A longer maximum history length may increase classification performance at the expense of longer training time. Tables 5.12 and 5.13 show the results for classification using a maximum history length of 2 and 3 (in contrast to the default length of 10 used in

[8]A β parameter of 1 is designed to give 'equal' weighting to precision and recall, but the interpretation is not completely clear.

[9]For brevity, the detail tables for the results on pruning and smoothing are not included in the current work.

Table 5.11: ePST Local without Smoothing

These results use a smoothing pseudocount of 0 rather than the default (0.01). The following data are obtained by 5-fold cross-validation over the given training sets. A length of 40 amino acids was used for the scoring window in each case except where noted otherwise.

Protein Class	Performance			
pkinase Pfam	Precision 0.072	Recall 0.072	Accuracy 0.973	Coverage 1.000
metallothio Pfam	Precision 0.077	Recall 0.077	Accuracy 0.997	Coverage 1.000
mixed Pfams	Precision 0.062	Recall 0.062	Accuracy 0.970	Coverage 1.000
K^+ Channels	Precision 0.610	Recall 0.610	Accuracy 0.805	Coverage 1.000
Gram- Subcell	Precision 0.519	Recall 0.519	Accuracy 0.805	Coverage 1.000
GO with SwissProt 1/4	Precision 0.361	Recall 0.361	Accuracy 0.750	Coverage 1.000

Table 5.12: ePST Local with Shorter History Length

These results use a maximum history length of 2 rather than the default (10). The following data are obtained by 5-fold cross-validation over the given training sets. A length of 40 amino acids was used for the scoring window in each case except where noted otherwise.

Protein Class	Performance			
pkinase Pfam	Precision 0.379	Recall 0.379	Accuracy 0.982	Coverage 1.000
metallothio Pfam	Precision 0.923	Recall 0.923	Accuracy 1.000	Coverage 1.000
mixed Pfams	Precision 0.430	Recall 0.430	Accuracy 0.982	Coverage 1.000
K^+ Channels	Precision 0.961	Recall 0.961	Accuracy 0.981	Coverage 1.000
Gram- Subcell	Precision 0.789	Recall 0.789	Accuracy 0.915	Coverage 1.000
GO with SwissProt 1/4	Precision 0.518	Recall 0.518	Accuracy 0.811	Coverage 1.000

Table 5.13: ePST Local with Shorter History Length

These results use a maximum history length of 3 rather than the default pruning (10). The following data are obtained by 5-fold cross-validation over the given training sets. A length of 40 amino acids was used for the scoring window in each case except where noted otherwise.

Protein Class	Performance			
pkinase Pfam	Precision 0.637	Recall 0.637	Accuracy 0.989	Coverage 1.000
metallothio Pfam	Precision 0.795	Recall 0.795	Accuracy 0.999	Coverage 1.000
mixed Pfams	Precision 0.673	Recall 0.673	Accuracy 0.990	Coverage 1.000
K^+ Channels	Precision 0.961	Recall 0.961	Accuracy 0.981	Coverage 1.000
Gram- Subcell	Precision 0.798	Recall 0.798	Accuracy 0.918	Coverage 1.000
GO with SwissProt 1/4	Precision 0.652	Recall 0.652	Accuracy 0.864	Coverage 1.000

these experiments). The importance of a history length can be seen in these results. Although longer history length is typically beneficial, there are some cases where performance is better with a shorter history length (as with the *Gram- Subcell* data set). This results correlates with the idea that much of a protein's subcellular location is determined by short regions of 'signaling' sequence. It was also found that there are diminishing returns as the maximum history length increases. A history length of 20 does not give any appreciable difference in performance compared to the default length of 10 (data not shown). This is likely because the parameters which use a history length beyond length 10 are very sparse and are not used very often in prediction.

It is apparent that the parameters of ePSTs have much to do with the performance on various data sets. The results shown here do not represent an exhaustive parameter search. These result do show that PSTs show some promise as predictors under certain circumstances where other methods fail. ePSTs may be useful where similar sequences cannot be found by alignment-based methods but this is difficult to verify since many of the data sets that we obtain for testing have largely been labeled through sequence similarity as determined by sequence alignment. PSTs (bPSTs or ePSTs) and HMMs can be helpful when no existing pattern (such those from Pfam or PROSITE) is known which corresponds to the particular protein family. HMMs, however, do not perform well with large or heterogeneous groups of proteins where a reasonable multiple sequence alignment is not possible. Under these circumstances, ePSTs are likely to have better classification performance than previous methods.

5.5 Comparison of Time Requirements

Although we usually want to use the technique with the best classification performance, this is not always possible due to limited resources. Tradeoffs between classification performance and time requirements may be acceptable in some cases. ePSTs are theoretically far more computationally efficient than the other techniques examined in this work. The following results compare the practical time requirements of each method. This information can be considered together with the information about classification performance in the previous section in order to select the best classification method for a particular protein function prediction task.

Comparisons were done for only the BLAST, HMM, and PST methods. Proteome Analyst is based on BLAST and has basically the same time requirements with some additional machine learning time. The use of the Pfam database is essentially the same as using multiple HMMs. As such, the use of Proteome Analyst and the scanning of the Pfam and PROSITE databases were excluded from this comparison.

Table 5.14: Comparison of Training Time Requirements Dependent on Training Set Size

All times are in seconds.

training seqs	BLAST formatdb	HMM clustalw	HMM hmmbuild	bPST train	ePST pst
10	0.02	0.63	0.46	23.86	0.01
20	0.02	2.06	0.50	93.76	0.02
30	0.01	4.15	0.47	210.65	0.03
40	0.00	7.13	0.42	6.35	0.04
50	0.02	10.33	0.38	9.62	0.05
60	0.01	14.26	0.42	13.51	0.07
70	0.02	19.73	0.49	18.74	0.07
80	0.02	26.10	0.50	11.69	0.08
90	0.01	30.98	0.49	14.01	0.10
100	0.02	38.52	0.52	18.15	0.11
200	0.02	148.52	0.74	28.05	0.23
300	0.03	327.24	1.31	39.44	0.35
400	0.02	586.77	2.19	55.29	0.48
500	0.05	932.79	3.43	66.30	0.64
600	0.04	1334.39	4.79	76.70	0.79
700	0.03	1875.28	5.98	89.10	0.93
800	0.04	2438.56	7.22	105.79	1.06
900	0.04	3255.01	9.92	115.97	1.20
1000	0.05	3876.15	6.78	126.36	1.33

For the following results, each of the programs was run using the same parameter setting that were used in the comparisons of classification performance (except where noted). The sequences used in each case were simulated sequences from the protein kinase Pfam family (*pkinase*, PF00069). They were randomly generated from the Pfam HMM using the `hmmemit` program and have an average length of about 280 amino acids.

The training time requirements of each classification method were compared across various numbers of training sequences. Since prediction time may be dependent on both the size of the testing set *and* the size of the training instance(s), the testing (prediction) time requirements were compared for various sizes of both sets.

The dependency of the training time for each method based on the size of the training set is shown in Table 5.14 and Figure 5.5. The dependency of prediction

Figure 5.5: Comparison of Training Time Requirements Dependent on Training Set Size

Data from Table 5.14 is plotted in this chart. All times are in seconds.

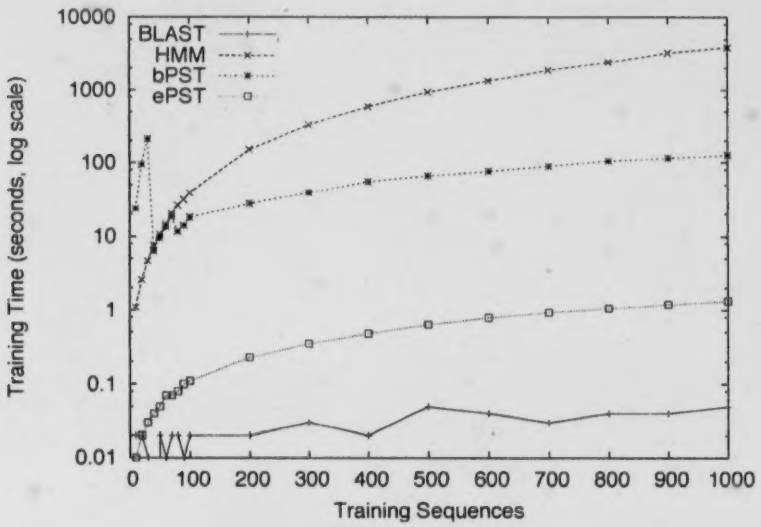


Table 5.15: Comparison of Prediction Time Requirements Dependent on Training Set Size

All times are in seconds.

training seqs	testing seqs	BLAST blastall	HMM hmmpfam	bPST predict	ePST pst
100	1000	112.22	15.71	0.39	1.01
200	1000	218.41	14.42	0.29	1.27
300	1000	316.40	16.44	0.29	1.35
400	1000	406.49	15.18	0.35	1.45
500	1000	496.83	16.77	0.33	1.46
600	1000	521.57	14.84	0.37	1.52
700	1000	551.37	21.99	0.33	1.46
800	1000	576.70	18.77	0.34	1.50
900	1000	604.73	21.33	0.33	1.49
1000	1000	631.92	24.16	0.33	1.43

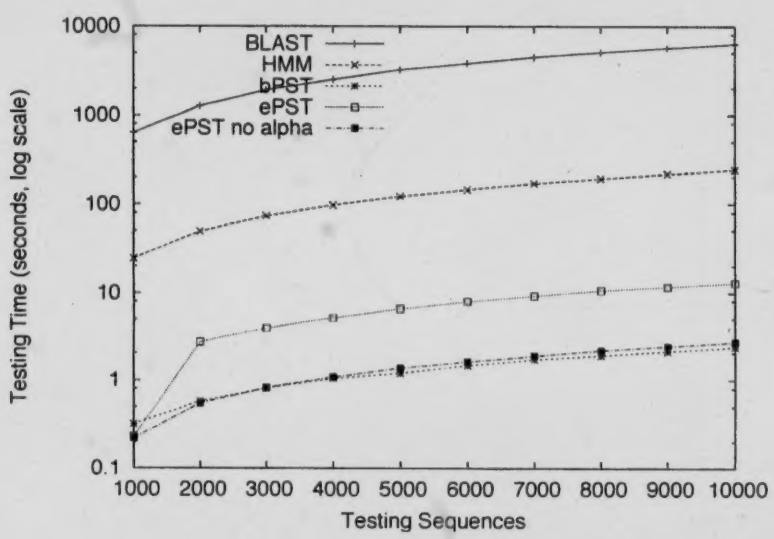
Table 5.16: Comparison of Prediction Time Requirements Dependent on Test Set Size

All times are in seconds.

training seqs	testing seqs	BLAST blastall	HMM hmmpfam	bPST predict	ePST pst	ePST no α
1000	1000	635.19	24.43	0.32	0.23	0.22
1000	2000	1273.76	48.88	0.58	2.65	0.55
1000	3000	1919.37	73.51	0.81	3.94	0.82
1000	4000	2545.99	97.02	1.05	5.20	1.09
1000	5000	3251.30	122.63	1.20	6.63	1.36
1000	6000	3855.68	146.05	1.45	7.96	1.61
1000	7000	4499.40	170.81	1.71	9.29	1.87
1000	8000	5096.30	193.47	1.90	10.65	2.14
1000	9000	5709.05	218.05	2.09	11.68	2.40
1000	10000	6350.49	242.77	2.33	12.94	2.68

Figure 5.6: Comparison of Testing Time Requirements Dependent on Testing Set Size

The data from Table 5.16 is plotted in this chart. All times are in seconds.



time based on the size of the training set is shown in Table 5.15. The dependency of prediction time based on the size of the test set is shown in Table 5.16 and Figure 5.6.

It can be seen that BLAST nearest-neighbor has essentially no training time. The 'classifier' is the set of training sequences. Considerable time is spent in the prediction phase finding the nearest neighbor (the top BLAST hit). A BLAST nearest-neighbor classifier is a relatively slow prediction method when compared to the others tested. The testing time seems to increase linearly with both the number of training and testing sequences.

HMMs require much more training time than BLAST. The training time is almost entirely due to the multiple sequence alignment. Unfortunately, the time required for this alignment increases more than linearly with the size of the training set. This characteristic explosion in processing time for multiple sequence alignment makes HMMs much less attractive for large classes of proteins when speed is required. The prediction times for HMMs, however, show little dependency on the size of the training set, so if the same HMM will be used many times (as those in the Pfam database are used) the expense of training the HMM one time may be acceptable. The prediction time for matching against HMMs is much less than BLAST for a single HMM. The prediction time for matching against the entire Pfam database, however, would be much more expensive since it contains more than 7300 families. Because each HMM is a different size, each one will require a different amount of time to run (up to 10X difference), but these results suffice to give some intuition of the time that would required if we were to use the entire Pfam database (possibly using a machine learning classifier for a prediction based on the Pfam matches).

The training time for bPSTs increases much more slowly than that of HMMs[10]. There is an interesting non-monotonic increase in the training time that appears to be dependent on the minimum probability required for placing a history in the tree (P_{min}). In Table 5.14 and Figure 5.14 two unexpected drops can be seen in the bPST training time – at 40 and 80 sequences. When using less than 40 training sequences with the published parameters ($P_{min} = 0.0001$) the training time is greater than for training with more than 1000 sequences. To examine this effect the bPST training was repeated where the parameter P_{min} was doubled and halved. The results of this testing can be seen in Table 5.17. Even though the value of P_{min} significantly changes the training time for bPSTs, that time would still be expected to be less than required for clustalw (it will increase more slowly for larger values) and greater than that required for ePSTs.

When using the default parameters, the testing time for bPSTs is very fast and does not appear to depend upon the size of the training set. This is likely because

[10]It should be noted that HMMs and PSTs may both be updated – a new sequence may be added to the family without recontructing the entire model from the start. This kind of updating is not considered in the current analysis.

Table 5.17: Non-Monotonic Increase in bPST Training Time Requirements Dependent on Training Set Size

All times are in seconds.

training seqs	bPST $P_{min} = 0.00020$	bPST $P_{min} = 0.00010$	bPST $P_{min} = 0.00005$	ePST pst
10	23.57	23.86	23.57	0.01
20	1.49	93.76	93.04	0.02
30	3.56	210.65	205.17	0.03
40	2.66	6.35	353.39	0.04
50	4.58	9.62	570.89	0.05
60	4.19	13.51	824.43	0.07
70	5.51	18.74	1067.78	0.08
80	5.25	11.69	23.98	0.09
90	7.06	14.01	30.46	0.11

the agressive pruning of bPSTs greatly limits the size of the tree regardless of the size of the training set. A check of the bPST predictions across the tests shown revealed that the average history length used was about 3 amino acids, regardless of the training set size (with the exception of the smallest training set of 10 sequences, for which the average was a little more than 2). This very short average history (despite the maximum history length of 20) likely allows the majority of the tree to fit in memory cache – and much faster memory access times. This characteristic does not change the theoretical attributes of the algorithm but may greatly contribute to the very fast prediction times for bPSTs.

The ePSTs have essentially the same classification performance as bPSTs. The small differences observed in classification performance are likely due to differences in pruning and smoothing which with experimentation could be better optimized for both algorithms. As classification performance is comparable for the two methods, differences in compute time are of interest.

The training time for ePSTs is much faster than for bPSTs (roughly 100x faster for 1000 training sequences). This difference in training time makes ePSTs more suitable to high-throughput protein function prediction tasks where many classifiers must be trained for many protein classes.

Surprisingly, the efficiency of ePSTs for prediction does not exceed that of bPSTs. This was examined and found to be due to differences in pruning and smoothing. A significant portion of the prediction time for ePSTS was used for calculating the backoff probabilities (specifically the value α). When the calculation of the

Table 5.18: Dependence of ePST Testing Time on Maximum History Length

All times are in seconds. In all of the runs shown below, the α calculation is made. The maximum history length L is shown for each run.

training seqs	testing seqs	ePST $L=0$	ePST $L=1$	ePST $L=2$	ePST $L=3$	ePST $L=10$	ePST $L=20$
1000	1000	0.02	0.02	0.06	0.21	0.22	0.21
1000	2000	0.04	0.05	0.11	1.31	2.60	2.59
1000	3000	0.06	0.07	0.17	1.97	3.89	3.89
1000	4000	0.08	0.10	0.22	2.62	5.21	5.19
1000	5000	0.10	0.12	0.28	3.25	6.49	6.48
1000	6000	0.13	0.15	0.35	3.94	7.78	7.78
1000	7000	0.15	0.17	0.41	4.59	9.06	9.07
1000	8000	0.17	0.20	0.47	5.24	10.38	10.36
1000	9000	0.19	0.22	0.51	5.87	11.64	11.63
1000	10000	0.21	0.25	0.57	6.55	12.94	13.10

backoff smoothing value α was removed (as shown in the last column of Table 5.16), the prediction calculation was much faster, although still not better than bPSTs. A second reason that ePSTs were not faster than bPSTs for prediction is related to the size of the tree. Although the theoretical analysis of the linear time algorithm suggests that this should not be the case, practical processing considerations have some influence. A small memory footprint for the tree may allow most of the memory access to occur in the on-chip memory cache, which is much more efficient than main memory access. Some additional experimental results demonstrate the effect of the size of the tree's memory footprint on the prediction speed.

Tables 5.18 and 5.19 and Figure 5.7 show some results for experiments with various values for the maximum history length L. Table 5.18 shows these results with the calculation of the backoff smoothing value α and Table 5.19 shows the same results without the expensive α calculation. It can be seen that for history lengths of 3 (matching the effective history length of the bPST with the published settings) or less, the ePST algorithm calculates the prediction probabilities much more quickly than bPSTs. Surprisingly, the calculation of the backoff value α has very little effect on the speed of the calculation for trees of these sizes. It appears from this preliminary investigation that the ePST tree can be mostly contained in the memory cache of the test system for maximum history lengths of 3 or less given this training data. It is likely that bPSTs with less stringent pruning criteria would show similarly slower prediction times with larger trees (as they extend beyond memory

Figure 5.7: Comparison of Testing Time Requirements Dependent on Testing Set Size and Tree Size

Data from Tables 5.16 and 5.18 is plotted in this chart. All times are in seconds. *The default value for L is 10 (including results found in Table 5.16) unless specified.

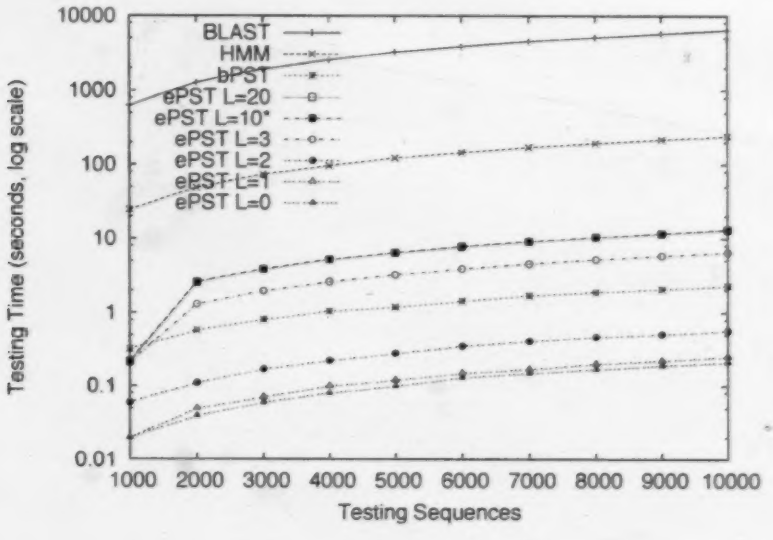


Table 5.19: Dependence of ePST Testing Time on Maximum History Length without the α Calculation

All times are in seconds. In all of the runs shown below, no α calculation is made. The maximum history length L is shown for each run.

training seqs	testing seqs	ePST $L = 0$	ePST $L = 1$	ePST $L = 2$	ePST $L = 3$	ePST $L = 10$	ePST $L = 20$
1000	1000	0.02	0.03	0.06	0.21	0.21	0.21
1000	2000	0.05	0.05	0.12	0.44	0.54	0.54
1000	3000	0.07	0.08	0.17	0.67	0.83	0.81
1000	4000	0.10	0.10	0.23	0.89	1.09	1.08
1000	5000	0.12	0.13	0.30	1.12	1.35	1.36
1000	6000	0.14	0.16	0.37	1.34	1.60	1.65
1000	7000	0.17	0.18	0.41	1.56	1.88	1.89
1000	8000	0.20	0.21	0.51	1.79	2.14	2.19
1000	9000	0.22	0.23	0.52	2.02	2.40	2.46
1000	10000	0.24	0.26	0.61	2.23	2.66	2.67

cache). Further work is required to confirm these assertions. It has been noted earlier that the ePST with a maximum history length of $L = 3$ still has better classification performance than bPSTs on the majority of the test sets (the *metallothio* Pfam family and the K^+ *Channels* being the exceptions for which ePSTs are only slightly behind). These results confirm that the ePST algorithm is much faster than the bPST algorithm as suggested by the theoretical analysis. They also warn, however, that implementation details and parameter selection can have a large influence on computational performance.

5.6 Summary

These results demonstrate the time requirements of various algorithms for protein function prediction. Although it is clear that PSTs are the most efficient methods for training and prediction, they do not have the best classification performance for all tasks. The ideal classifier for protein function prediction is very dependent on factors such as

- the type of pattern which best defines the particular protein class,
- the classification performance required,

- the computational resources available,
- the time available,
- the number of training sequences to be used, and
- the number of predictions to be made.

For each classification task the choice of the best tool will require some knowledge of the conditions under which it will be used.

Chapter 6

Conclusion

6.1 Discussion of Results

Francis Crick[1] has restated the Central Dogma in the following way: 'Once information has passed into protein, it cannot get out again.' The current work has shown that once information has passed into protein *sequence*, at least, there is still much information which can be extracted from it.

Biological sequences contain a plethora of information about biological systems. The content of this work has illustrated that the amino acid sequences of proteins provide sufficient information to predict many protein properties, including function. Many tools are available for protein function prediction based on sequence information. Each takes advantage of different types of sequence patterns and is appropriate for different prediction tasks. No single method excels at all tasks. It is crucial that those who intend to computationally analyze protein sequences select the best tool for the particular task of interest.

A protein function prediction tool for automated and high-throughput protein function prediction should be

- accurate,
- efficient,
- flexible, and
- transparent.

The properties of several methods have been examined with respect to these characteristics.

[1]Francis Crick passed away on July 28, 2004 — about a month before the submission of this work. As biological sequences are threaded through the machinery of life, so is his work threaded through the study of it.

Alignment-based methods have been shown to be very accurate predictors for a wide range of tasks. Nearest neighbor prediction based on BLAST scores also provides biological intuition for the prediction function based on alignments. BLAST is very efficient compared to earlier algorithms, but still requires significant computational resources. One of the key problems with alignment-based methods is the inability to predict when similar sequences cannot be found (low coverage). Proteome Analyst improves on the BLAST nearest neighbor predictor by utilizing machine learning and the information from well-annotated protein databases. Although the interpretability of the prediction is also excellent due to an explanation facility, the time requirements of the method are not improved. The problem of coverage is reduced, but not removed.

Hidden Markov models are very accurate predictors of conserved protein families. When a multiple sequence alignment and corresponding model can be made for a protein family, HMMs perform better than sequence alignments or PSTs. They do not appear to be able to represent well a protein class composed of several heterogeneous subclasses. The training time requirements for large classes can be prohibitively expensive. The prediction time for a single protein family is better than for using nearest-neighbor sequence alignment. HMM predictions, although not as simple to understand as alignments, can give also some intuition about matches to the conserved pattern in the sequence family.

Pattern databases, some of which include HMMs, are useful for a variety of tasks. They are very useful when the desired protein class has been studied previously and a corresponding pattern is found in the database. Even when the particular group of proteins is not identified by a particular pattern in the databases, combinations of other patterns may be used effectively in concert with a machine learning classifier. The time required to match against these databases exceeds both the time required to match a single HMM and the time required to find the most similar sequences by alignment.

Probabilistic suffix trees show some promise as more efficient and flexible alternatives to HMMs. They can be used to predict much more efficiently than any previous method. They are an alignment free method and have much lesser training time requirements. The coverage problems seen with alignments do not affect PSTs but the predictions of PSTs are not as reliable as alignments where similar sequences can be found. They approach the accuracy of HMMs for some protein families but not all. They can, however, be used for larger protein classes for which HMMs cannot. PSTs are conceptually more simple than HMMs and the results of their predictions, although very similar to the predictions of HMMs, may be easier to interpret. The current work introduced an efficient implementation of PSTs that can be trained in linear time with respect to the total length of the training sequences. The training time is more than one hundred times faster than the previous implementations (bPSTs). The prediction time for ePSTs is linear with respect to

the length of the test sequence. This can lead to much faster prediction times than bPSTs. In practice, however, the expense of the improved backoff smoothing calculation and the slightly larger memory footprint due to less aggressive pruning may lead to a situation in which a short bPST may still be found to be faster than ePSTs.

Overall, PSTs show promise for high-throughput and automatic protein function prediction. Although alignment-based methods and HMMs still give the best classification performance for most conditions, there are circumstances under which neither BLAST nor HMMs can be easily applied. PSTs will greatly enhance performance in these cases.

6.2 Future Work

The current work points to a variety of future work which may be done in protein function prediction. Future work includes further experimentation with current PST implementations, improvement to PST implementations, and development of new methods which may be able to take advantage of some lessons learned from PSTs.

6.2.1 Experimentation with PSTs

PSTs have been show to approximate the performance of HMMs for many classes, but have not been investigated as thoroughly as HMMs for protein function prediction. Further investigations into selection of pruning and smoothing criteria may improve classification performance. It will be particularly important that the selection of better performing criteria be automated in some way so that non-expert users may still find PSTs useful. More sophisticated pruning might also make better use of the bias-variance tradeoff.

We have seen that PSTs may be used for reasonable classification performance on both large and small protein classes. This leads naturally to applications in hierarchical classification where both large and small classes must be classified. Combinations of PSTs that represent protein classes and subclasses may also be very useful in concert. Because of their efficiency they are well suited to classification of large databases of protein sequences with numerous classes. Future work may focus especially on Gene Ontology prediction for newly sequenced proteins – especially those that have no good sequence alignments against current protein databases. Groups of PSTs may also be useful for many other non-hierarchical classification tasks.

A PST database would be useful for classes of proteins that are well represented by PST models. Such a database would allow very efficient scanning and prediction of established protein classes.

Because of the very efficient training and prediction capabilities of PSTs, they

may be useful as heuristic filters for other techniques such as HMMs. This would be analogous to the manner in which BLAST alignments are started from promising seeds that are heuristically and efficiently selected and then investigated with more computationally expensive and accurate calculations. Such a tool might be particularly useful for accelerating searches of HMM databases such as Pfam.

Because of their flexibility and efficiency, PSTs may be included as tools for integrated methods such as Proteome Analyst and InterProScan without greatly affecting the time requirements of these systems. When PSTs are used in conjunction with other methods such as HMMs and BLAST, the other methods will be expected to dominate the computational time.

PSTs may also have further application to other sequence analysis tasks such as processing of natural language texts.

6.2.2 Improvements to PSTs

The PST model has been shown in this context as a generative model. Work is in progress with regard to representing both positive and negative training data in a single ePST. The tree would then be pruned and smoothed to select the parameters which best aid in discriminating between the classes. The work of using PSTs as discriminative models could also be extended to multi-class problems. Some preliminary work has begun on this topic with VMMs[85].

In this work we examined some improvements on pruning, smoothing, and backoff. It is apparent from the results that pruning and smoothing can have a significant effect on classification performance and it is possible that other ideas from other Markov chains such as deleted interpolation be done more efficiently with PSTs. Some parameter selection using EM or Gibbs sampling might also be of interest.

It was also seen that the calculation of the backoff smoothing value α was quite expensive in ePSTs. A simpler approximation to this value which could be cached in the tree at build time would improve the prediction speed of the ePST algorithms.

This work has shown that a smaller memory footprint of the PST drastically increases performance. A decrease in the memory footprint of the ePST application may yield practical performance benefits.

The question of introducing more flexibility into the PST model is also open. Allowing mismatches has been examined to some degree[37] but further investigation is warranted.

The local predictions of Sun and Deogun[89] were not fully examined in this work. It is likely that those ideas be implemented and improved upon in ePSTs. The incorporation of bidirectional prediction may improve local PST prediction of pattern occurrences as well.

Probabilistic suffix trees do not currently have a good visualization which allows

users to have some global representation of the protein family represented by the tree. It might be particularly useful to visually present and highlight portions of the tree where predictions either depart widely from their expected values or contribute highly to a given classification.

6.2.3 New Methods Related to PSTs

String kernels for classification are a promising area of research for protein function. They provide a discriminative model which may be used directly on sequences. Because of the rich sequence information which can be calculated efficiently with suffix trees, they are a natural fit for string kernel development[57]. The basic suffix tree implementation developed here can be used to test various string kernel architectures. The principles developed with PSTs and discriminative VMMs may also be useful in this domain. There may be an opportunity for developing a kernel similar to the Fisher kernel for HMMs[47].

As string kernels introduce the notion of similarity between sequences, suffix trees and PSTs may also be used to define a clustering model that may help automate the process of dealing with hierarchically related proteins.

6.3 Summary

This work has provided an in-depth analysis of the classification performance and time requirements of several techniques for protein function prediction. In addition, an efficient implementation of a relatively new pattern model, probabilistic suffix trees, was introduced. The tradeoffs between speed and accuracy were examined, as well as the circumstances for which each function prediction technique is best suited. Automatic and high-throughput protein function prediction is a task for which many tools are needed. A knowledge of the appropriate tool for each classification task is essential to success in computational analysis of proteins. Through better use of these sequence analysis tools we will be better able to take advantage of the large volume of available sequence information and organize it into profitable knowledge about living systems.

This work has demonstrated the effectiveness of current techniques for various high-throughput protein function prediction tasks based on sequence information. It has also shown that probabilistic suffix trees offer a method of greatly increasing the efficiency of these predictions.

Bibliography

[1] Wikipedia, the free encyclopedia. http://www.wikipedia.org, 2004.

[2] A Aho and M Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18:333–340, 1975.

[3] DG Altman and JM Bland. Statistics Notes: Diagnostic tests 3: receiver operating characteristic plots. *BMJ*, 309(6948):188–, 1994.

[4] S Altschul, W Gish, W Miller, EW Myers, and D Lipman. A basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

[5] SF Altschul, TL Madden, AA Schaffer, J Zhang, Z Zhang, W Miller, and DJ Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25(17):3389–3402, 1997.

[6] CAF Andersen and S Brunak. Representation of protein sequence information by amino acid subalphabets. *AI Magazine*, 25:97–104, 2004.

[7] A Apostolico and G Bejerano. Optimal amnesic probabilistic automata or how to learn and classify proteins in linear time and space. *Journal of Computational Biology*, 7(3/4):381–393, 2000.

[8] EM Zdobnov R Apweiler. InterProScan - an integration platform for the signature-recognition methods in InterPro. *Bioinformatics*, 17(9):847–8, 2001.

[9] R Apweiler, A Bairoch, CH Wu, WC Barker, B Boeckmann, S Ferro, E Gasteiger, H Huang, R Lopez, M Magrane, MJ Martin, DA Natale C O'Donovan, N Redaschi, and LS Yeh. UniProt: the Universal Protein knowledgebase. *Nucleic Acids Research*, 32:D115–D119, 2004.

[10] TL Bailey and C Elkan. Fitting a mixture model by expectation maximization to discover motifs in biopolymers. In *Proceedings of the Second International Conference on Intelligent Systems for Molecular Biology*, pages 28–36. AAAI Press, Menlo Park, California, 1994.

[11] TL Bailey and M Gribskov. Combining evidence using p-values: application to sequence homology searches. *Bioinformatics*, 14:48–54, 1998.

[12] TL Bailey and M Gribskov. Methods and statistics for combining motif match scores. *Journal of Computational Biology*, 5(2):211–222, 1998.

[13] A Bateman, E Birney, L Cerruti, R Durbin, L Etwiller, SR Eddy, S Griffiths-Jones, KL Howe, M Marshall, and EL Sonnhammer. The Pfam protein families database. *Nucleic Acids Research*, 30(1):276–280, 2002.

[14] G Bejerano. Personal communication. 2004.

[15] G Bejerano. Algorithms for variable length markov chain modeling. *Bioinformatics Applications Note*, 20(5):788–789, 2004.

[16] G Bejerano and G Yona. Modeling protein families using probabilistic suffix trees. In *The Third Annual International Conference on Computational Molecular Biology (RECOMB'99)*, pages 15–24, Lyon, France, April 1999. ACM press.

[17] G Bejerano and G Yona. Variations on probabilistic suffix trees: statistical modeling and prediction of protein families. *Bioinformatics*, 17(1):23–43, 2001.

[18] HM Berman, J Westbrook, Z Feng, G Gilliland, TN Bhat, H Weissig, IN Shindyalov, and PE Bourne. The Protein Data Bank. *Nucleic Acids Research*, 28:235–242, 2000.

[19] B Boeckmann, A Bairoch, R Apweiler, MC Blatter, A Estreicher, E Gasteiger, MJ Martin, K Michoud, C O'Donovan, I Phan, S Pilbout, and M Schneider. The SWISS-PROT protein knowledgebase and its supplement TrEMBL in 2003. *Nucleic Acids Research*, 31:365–370, 2003.

[20] GEP Box. Robustness is the strategy of scientific model building. In RL Launer and GN Wilkinson, editors, *Robustness in Statistics*, pages 201–236, New York, NY, 1979. Academic Press.

[21] RS Boyer and JS Moore. A fast string searching algorithm. *Communications of the ACM*, 20:762–772, 1977.

[22] A Brazma, I Jonassen, I Eidhammer, and D Gilbert. Approaches to the automatic discovery of patterns in biosequences. *Journal of Computational Biology*, 5(2):279–305, 1998.

[23] P Buhlmann and AJ Wyner. Variable length markov chains. *Annals of Statistics*, (2):480–513, 1999.

[24] C Burge and S Karlin. Prediction of complete gene structures in human genomic dna. *Journal of Molecular Biology*, 268:78–94, 1997.

[25] A Califano. SPLASH: structural pattern localization analysis by sequential histograms. *Bioinformatics*, 16(4):341–357, 2000.

[26] J Cameron and W Wisher. Terminator 2: Judgement Day. Columbia TriStar Pictures, 1991.

[27] E Camon, D Barrell, V Lee, E Dimmer, and R Apweiler. The Gene Ontology Annotation (GOA) Database - An integrated resource of GO annotations to the UniProt Knowledgebase. *In Silico Biology*, 2003.

[28] The Gene Ontology Consortium. Gene Ontology: tool for the unification of biology. *Nature Genet*, (25):25–29, 2000.

[29] FHC Crick. On protein synthesis. In *Symposium Society Experimental Biology*, number 12, pages 138–163, 1958.

[30] N Cristianini. Support vector and kernel methods for learning. Tutorial in the Eighteenth International Conference on Machine Learning (ICML 2001), June 2001.

[31] MO Dayhoff, RM Schwartz, and BC Orcutt. *A model of evolutionary change in proteins. Matrices for detecting distant relationships*, volume 5, pages 345–358. National Biomedical Research Foundation, Washington DC, 1978.

[32] AL Delcher, D Harmon, S Kasif, O White, and SL Salzberg. Improved microbial gene identification with GLIMMER. *Nucleic Acids Research*, 27(23):4636–4641, 1999.

[33] B Dorohonceanu and CG Nevill-Manning. Accelerating protein classification using suffix trees. In *The Eighth International Conference on Intelligent Systems for Molecular Biology*, pages 128–133, La Jolla / San Diego, CA, August 2000. AAAI Press.

[34] R Durbin, S Eddy, A Krogh, and G Mitchison. *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge University Press, 1998.

[35] SR Eddy. HMMER: Profile hidden markov models for biological sequence analysis. http://hmmer.wustl.edu, 2001.

[36] R Eisner. Personal communication. 2004.

[37] E Eskin, WN Grundy, and Y Singer. Protein family classification using sparse markov transducers. In *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*, pages 134–135, August 2000.

[38] A Gattiker, E Gasteiger, and A Bairoch. ScanProsite: a reference implementation of a prosite scanning tool. *Applied Bioinformatics*, 1:107–108, 2002.

[39] O Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162:705–708, 1982.

[40] D Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997.

[41] J Han and M Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2000.

[42] T Hastie, R Tibshirani, and J Friedman. *The Elements of Statistical Learning*. Springer, 2001.

[43] S Henikoff and JG Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences*, 19:6565–6572, 1991.

[44] S Henikoff and JG Henikoff. Automated assembly of protein blocks for database searching. *Nucleic Acids Research*, 19:6565–6572, 1991.

[45] S Henikoff and JG Henikoff. Amino Acid Substitution Matrices from Protein Blocks. *PNAS*, 89(22):10915–10919, 1992.

[46] DS Hirschberg. A linear space algorithm for computing maximal common subsequences. *Communications of the ACM*, 18(6):341–343, 1975.

[47] T Jaakkola, M Diekhans, and D Haussler. A discriminative framework for detecting remote protein homologies. *Journal of Computational Biology*, 7(1,2):95–114, 2000.

[48] I Jonassen. Efficient discovery of conserved patterns using a pattern graph. *CABIOS*, 13:509–522, 1997.

[49] I Jonassen, JF Collins, and DG Higgins. Finding flexible patterns in unaligned protein sequences. *Protein Science*, 4:1587–1595, 1995.

[50] D Jurafsky and JH Martin. *Speech and Language Processing*. Prentice Hall, 2000.

[51] C Kermorvant and P Dupont. Improved smoothing for probabilistic suffix trees seen as variable order markov chains. In *Proceedings of the 13th European Conference on Machine Learning*, pages 185–194, Helsinki, Finland, 2002.

[52] DE Knuth, JH Morris, and VB Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6:323–350, 1977.

[53] R Kohavi and F Provost. Glossary of terms. *Machine Learning*, 30:271–274, 1998.

[54] S Kurtz, A Phillippy, AL Delcher, M Smoot, M Shumway, C Antonescu, and SL Salzberg. Versatile and open software for comparing large genomes. *Genome Biology*, 5(R12), 2004.

[55] RA Laskowski. PDBsum: summaries and analyses of PDB structures. *Nucleic Acids Research*, 29:221–222, 2001.

[56] CE Lawrence, SF Altschul, MS Boguski, JS Liu, AF Neuwald, and JC Wootton. Detecting subtle sequence signals: A gibbs sampling strategy for multiple alignment. *Science*, 262(5131):208–214, 1993.

[57] C Leslie, E Eskin, A Cohen, J Weston, and W Noble. Mismatch string kernels for discriminative protein classification. *Bioinformatics*, 20(4):467–76, 2004.

[58] B Lewin. *Genes VI*. Oxford University Press, Oxford, 1997.

[59] B Li and WJ Gallin. VKCDB: Voltage-gated potassium channel database. *BMC Bioinformatics*, 5(3), 2004.

[60] C Li, editor. *Biochemical Nomenclature and Related Documents*. Portland Press, second edition edition, 1992.

[61] DJ Lipman and WR Pearson. Rapid and sensitive protein similarity searches. *Science*, 227:1435–1441, 1985.

[62] Z Lu. Predicting protein sub-cellular localization from homologs using machine learning algorithms. Master's thesis, Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada, 2003.

[63] Z Lu, D Szafron, R Greiner, P Lu, D Wishart, B Poulin, J Anvik, C Macdonell, and R Eisner. Predicting sub-cellular localization using machine-learned classifiers in proteome analyst. *Bioinformatics*, 20:547–556, 2004.

[64] LB Lusted. Signal detectability and medical decision making. *Science*, 171:1217–1219, 1971.

[65] Veli Makinen. *Parameterized Approximate String Matching and Local-Similarity Point-Pattern Matching*. PhD thesis, University of Helsinki, Finland, 2003.

[66] AA Markov. Essai d'une recherche statistique sure le texte du roman 'eugene onegin' illustrant la liaison des epreuve en chain (example of a statistical investigation of the test of 'eugene onegin' illustrating the dependence between samples in chain). *Izvistia Imperatorskoi Akademii Nauk (Bulletin de l'Académie Impériale des Sciences de St.-Pétersbourg*, 7:153–162, 1913. English translation by Morris Halle, 1956.

[67] L Marsan and MF Sagot. Algorithms for extracting structured motifs using a suffix-tree with application to promoter and regulatory site consensus identification. *Journal of Computational Biology*, 7:345–360, 2001.

[68] EM McCreight. A space-economical suffix tree construction algorithm. *Journal of the ACM*, 23:262–272, 1976.

[69] EW Meyers and W Miller. Optimal alignments in linear space. *Computer Applications in Biosciences*, 4:11–17, 1988.

[70] T Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[71] NJ Mulder, R Apweiler, TK Attwood, A Bairoch, D Barrell, A Bateman, D Binns, M Biswas, P Bradley, P Bork, P Bucher, RR Copley, E Courcelle, U Das, R Durbin, L Falquet, W Fleischmann, S Griffiths-Jones, D Haft, N Harte, N Hulo, D Kahn, A Kanapin, M Krestyaninova, R Lopez, I Letunic, D Lonsdale, V Silventoinen, SE Orchard, M Pagni, D Peyruc, CP Ponting, JD Selengut, F Servant, CJA Sigrist, R Vaughan, and EM Zdobnov. The InterPro Database, 2003 brings increased coverage and new features. *Nucleic Acids Research*, (31):315–318, 2003.

[72] LR Murphy, A Wallqvist, and RM Levy. Simplified amino acid alphabets for protein fold recognition and implications for folding. *Protein Engineering*, 13(3):149–52, March 2000.

[73] AG Murzin, SE Brenner, T Hubbard, and C Chothia. SCOP: a structural classification of proteins database for the investigation of sequences and structures. *Journal of Molecular Biology*, 247:536–540, 1995.

[74] SB Needleman and CD Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, (48):443–453, 1970.

[75] H Ney, U Essen, and R Kneser. On structuring probabilistic dependencies in stochastic language modeling. *Computer Speech and Language*, 8:1–38, 1994.

[76] A Niccol. Gattaca. Columbia TriStar Studios, 1997.

[77] H Nielsen and A Krogh. Prediction of signal peptides and signal anchors by a hidden markov model. In *Proceedings of the Sixth International Conference on Intelligent Systems for Molecular Biology*, pages 122–130, Menlo Park, California, 1998. AAAI Press.

[78] D Pribnow. Nucleotide sequence of an rna polymerase binding site at an early t7 promoter. *Proceedings of the National Academy of Sciences*, (72):784–788, 1975.

[79] JR Quinlan. Induction of decision trees. In JW Shavlik and TG Dietterich, editors, *Readings in Machine Learning*. Morgan Kaufmann, 1990. Originally published in *Machine Learning* 1:81–106, 1986.

[80] I Rigoutsos and A Floratos. Combinatorial pattern discovery in biological sequences: the TEIRESIAS algorithm. *Bioinformatics*, 14(1), 1998.

[81] J Rissanen. A universal data compression system. *IEEE Transactions on Information Theory*, (5):656–664, 1983.

[82] D Ron, Y Singer, and N Tishby. The power of amnesia: Learning probabilistic automata with variable memory length. *Machine Learning*, 25(2-3):117–149, 1996.

[83] TD Schneider. Consensus sequence zen. *Applied Bioinformatics*, 1(3):111–119, 2002.

[84] CJ Sigrist, L Cerutti, N Hulo, A Gattiker, L Falquet, M Pagni, A Bairoch, and P Bucher. PROSITE: a documented database using patterns and profiles as motif descriptors. *Brief Bioinformatics*, 3:265–274, 2002.

[85] N Slonim, G Bejerano, S Fine, and N Tishby. Discriminative feature selection via multiclass variable memory markov model. In *International Conference on Machine Learning 2002*, 2002.

[86] TF Smith and MS Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[87] AJ Smola, P Bartlett, B Schölkopf, and D Schuurmans. *Advances in Large Margin Classifiers*. MIT Press, Cambridge, MA, 2000.

[88] ELL Sonnhammer, G von Heijne, and A Krogh. A hidden markov model for predicting transmembrane helices in protein sequences. In *Proceedings of the Sixth International Conference on Intelligent Systems for Molecular Biology*, pages 175–182, Menlo Park, California, 1998. AAAI Press.

[89] Z Sun and JS Deogun. Local prediction approach of protein classification using probabilistic suffix trees. In YPP Chen, editor, *Proc. Second Asia-Pacific Bioinformatics Conference (APBC2004)*, number 29 in Conferences in Research and Practice in Information Technology, pages 357–362, Dunedin, New Zealand, January 2004. Australian Computer Society.

[90] D Szafron, P Lu, R Greiner, D Wishart, Z Lu, B Poulin, R Eisner, J Anvik, and C Macdonell. Proteome Analyst - transparent high-throughput protein annotation: Function, localization and custom predictors. In *International Conference on Machine Learning Workshop on Machine Learning in Bioinformatics (ICML Workshop - Bioinformatics)*, pages 2–10, Washington, U.S.A., August 2003.

[91] D Szafron, P Lu, R Greiner, DS Wishart, B Poulin, R Eisner, Z Lu, J Anvik, C Macdonell, A Fyshe, and D Meeuwis. Proteome Analyst: custom predictions with explanations in a web-based tool for high-throughput proteome annotations. *Nucleic Acids Research*, 32(2):W365–371, 2004.

[92] JD Thompson, DG Higgins, and TJ Gibson. CLUSTALW: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22(22):4673–4680, November 1994.

[93] W Thompson, EC Rouchka, and CE Lawrence. Gibbs Recursive Sampler: finding transcription factor binding sites. *Nucleic Acids Research*, 31(13):3580–3585, 2003.

[94] E Ukkonen. On-line construction of suffix-trees. *Algorithmica*, 14:249–260, 1995.

[95] L Wall. Perl. www.perl.org, 2004.

[96] L Wang and T Jiang. On the complexity of multiple sequence alignment. *Journal of Computational Biology*, 1(4):337–348, 1994.

[97] P Weiner. Linear pattern matching algorithms. *Proceedings of the 14th IEEE Symposium on Switching and Automata Theory*, pages 1–11, 1973.

[98] IH Witten and E Frank. *Data mining: Practical machine learning tools and techniques with Java implementations.* Morgan Kaufmann, San Francisco, 2000.

[99] X Zhou, DK McClish, and NA Obuchowski. *Statistical Methods in Diagnostic Medicine.* Wiley, 2002.

Appendix A

Supplementary Results

Table A.1: Results for K^+ Channels, Majority Classifier

5-fold cross-validation results.

Protein Class	Results				Performance		
	TP	FN	FP	TN	Precision	Recall	Accuracy
Kv1	0	24	0	53	—	0.000	0.688
Kv2	0	17	0	60	—	0.000	0.779
Kv3	0	17	0	60	—	0.000	0.779
Kv4	0	19	0	58	—	0.000	0.753
OVERALL	0	77	0	231	—	0.000	0.750

Table A.2: Results for Gram- Subcell, Majority Classifier

5-fold cross-validation results.

Protein Class	Results				Performance		
	TP	FN	FP	TN	Precision	Recall	Accuracy
cytoplasm	2487	0	1473	0	0.628	1.000	0.628
extracellular	0	204	0	3756	—	0.000	0.948
inner membrane	0	606	0	3354	—	0.000	0.847
outer membrane	0	288	0	3672	—	0.000	0.927
periplasm	0	429	0	3531	—	0.000	0.892
OVERALL	2487	1527	1473	14313	0.628	0.620	0.848

Table A.3: Results for GO with SwissProt 1/4, Majority Classifier

5-fold cross-validation results.

Protein Class	Results				Performance		
binding (0005488)	TP	0	FN	12947	Precision	Recall	Accuracy
	FP	0	TN	12965	—	0.000	0.500
catalytic activity (0003824)	TP	15379	FN	0	Precision	Recall	Accuracy
	FP	10533	TN	0	0.594	1.000	0.594
hydrolase activity (0016787)	TP	0	FN	4347	Precision	Recall	Accuracy
	FP	0	TN	21565	—	0.000	0.832
lyase activity (0016829)	TP	0	FN	1444	Precision	Recall	Accuracy
	FP	0	TN	24468	—	0.000	0.944
metal ion binding (0046872)	TP	0	FN	2527	Precision	Recall	Accuracy
	FP	0	TN	23385	—	0.000	0.902
nucleic acid binding (0003676)	TP	0	FN	5335	Precision	Recall	Accuracy
	FP	0	TN	20577	—	0.000	0.794
nucleotide binding (0000166)	TP	0	FN	4054	Precision	Recall	Accuracy
	FP	0	TN	21858	—	0.000	0.844
oxidoreductase activity (0016491)	TP	0	FN	2732	Precision	Recall	Accuracy
	FP	0	TN	23180	—	0.000	0.895
signal transducer activity (0004871)	TP	0	FN	2028	Precision	Recall	Accuracy
	FP	0	TN	23884	—	0.000	0.922
structural molecule activity (0005198)	TP	0	FN	1986	Precision	Recall	Accuracy
	FP	0	TN	23926	—	0.000	0.923
transferase activity (0016740)	TP	0	FN	4488	Precision	Recall	Accuracy
	FP	0	TN	21424	—	0.000	0.827
transporter activity (0005215)	TP	0	FN	3604	Precision	Recall	Accuracy
	FP	0	TN	22308	—	0.000	0.861
OVERALL	TP	15379	FN	45492	Precision	Recall	Accuracy
	FP	10533	TN	239540	0.594	0.253	0.820

Table A.4: Results for K^+ Channels, BLAST Nearest-Neighbor

5-fold cross-validation results. A BLAST E-value threshold of 10 was used. Coverage is 1.0.

Protein Class	Results				Performance		
Kv1	TP 24	FN 0			Precision	Recall	Accuracy
	FP 0	TN 53			1.000	1.000	1.000
Kv2	TP 17	FN 0			Precision	Recall	Accuracy
	FP 0	TN 60			1.000	1.000	1.000
Kv3	TP 17	FN 0			Precision	Recall	Accuracy
	FP 0	TN 60			1.000	1.000	1.000
Kv4	TP 19	FN 0			Precision	Recall	Accuracy
	FP 0	TN 58			1.000	1.000	1.000
OVERALL	TP 60	FN 0			Precision	Recall	Accuracy
	FP 0	TN 231			1.000	1.000	1.000

Table A.5: Results for Gram- Subcell, BLAST Nearest Neighbor

5-fold cross-validation results. A BLAST E-value threshold of 10 was used. Coverage is 0.996.

Protein Class	Results						Performance		
cytoplasm	TP 2397	FN 86	NPP 4				Precision	Recall	Accuracy
	FP 146	TN 1317	NPN 10				0.943	0.964	0.938
extracellular	TP 153	FN 48	NPP 3				Precision	Recall	Accuracy
	FP 31	TN 3714	NPN 11				0.832	0.750	0.977
inner membrane	TP 506	FN 94	NPP 6				Precision	Recall	Accuracy
	FP 29	TN 3317	NPN 8				0.946	0.835	0.965
outer membrane	TP 255	FN 32	NPP 1				Precision	Recall	Accuracy
	FP 23	TN 3636	NPN 13				0.917	0.885	0.983
periplasm	TP 303	FN 124	NPP 2				Precision	Recall	Accuracy
	FP 39	TN 3480	NPN 12				0.886	0.706	0.955
OVERALL	TP 3614	FN 384	NPP 16				Precision	Recall	Accuracy
	FP 268	TN 15464	NPN 54				0.931	0.900	0.964

Table A.6: Results for GO with SwissProt 1/4, BLAST Nearest-Neighbor

5-fold cross-validation results. A BLAST E-value threshold of 10 was used. The following results were obtained by 1-nearest-neighbor classification on selected Gene Ontology categories over a random selection of 25912 sequences from the SwissProt database. 2- and 3-nearest-neighbor classifications gave nearly identical results. Coverage is 0.992.

Protein Class	Results TP	FN	NPP	Performance		
	FP	TN	NPN			
binding (0005488)	12227	612	108	Precision	Recall	Accuracy
	1016	11862	87	0.923	0.944	0.930
catalytic activity (0003824)	14960	357	62	Precision	Recall	Accuracy
	669	9731	133	0.957	0.973	0.953
hydrolase activity (0016787)	3940	387	20	Precision	Recall	Accuracy
	328	21062	175	0.923	0.906	0.965
lyase activity (0016829)	1334	107	3	Precision	Recall	Accuracy
	60	24216	192	0.957	0.924	0.986
metal ion binding (0046872)	2173	340	14	Precision	Recall	Accuracy
	332	22872	181	0.867	0.860	0.967
nucleic acid binding (0003676)	4825	468	42	Precision	Recall	Accuracy
	393	20031	153	0.925	0.904	0.959
nucleotide binding (0000166)	3945	104	5	Precision	Recall	Accuracy
	402	21266	190	0.908	0.973	0.973
oxidoreductase activity (0016491)	2503	212	17	Precision	Recall	Accuracy
	152	22850	178	0.943	0.916	0.978
signal transducer activity (0004871)	1732	261	35	Precision	Recall	Accuracy
	149	23575	160	0.921	0.854	0.977
structural molecule activity (0005198)	1861	101	24	Precision	Recall	Accuracy
	42	23713	171	0.978	0.937	0.987
transferase activity (0016740)	4153	323	12	Precision	Recall	Accuracy
	288	20953	183	0.935	0.925	0.969
transporter activity (0005215)	3200	379	25	Precision	Recall	Accuracy
	185	21953	170	0.945	0.888	0.971
OVERALL	56853	3651	367	Precision	Recall	Accuracy
	4016	244084	1973	0.934	0.934	0.968

Table A.7: Results for K^+ Channels, Proteome Analyst

Coverage is 1.000.

Protein Class	Results				Performance		
Kv1	TP 24	FN 0	FP 0	TN 53	Precision 1.000	Recall 1.000	Accuracy 1.000
Kv2	TP 17	FN 0	FP 0	TN 60	Precision 1.000	Recall 1.000	Accuracy 1.000
Kv3	TP 17	FN 0	FP 0	TN 60	Precision 1.000	Recall 1.000	Accuracy 1.000
Kv4	TP 19	FN 0	FP 0	TN 58	Precision 1.000	Recall 1.000	Accuracy 1.000
OVERALL	TP 60	FN 0	FP 0	TN 231	Precision 1.000	Recall 1.000	Accuracy 1.000

Table A.8: Results for Gram- Subcell, Proteome Analyst

Coverage is 1.000.

Protein Class	Results				Performance		
cytoplasm	TP 2436	FN 51	FP 3	TN 1470	Precision 0.999	Recall 0.979	Accuracy 0.986
extracellular	TP 198	FN 6	FP 70	TN 3686	Precision 0.739	Recall 0.971	Accuracy 0.981
inner membrane	TP 593	FN 13	FP 29	TN 3325	Precision 0.953	Recall 0.979	Accuracy 0.989
outer membrane	TP 281	FN 7	FP 15	TN 3656	Precision 0.949	Recall 0.976	Accuracy 0.994
periplasm	TP 418	FN 11	FP 48	TN 3483	Precision 0.897	Recall 0.974	Accuracy 0.985
OVERALL	TP 3926	FN 88	FP 165	TN 15621	Precision 0.960	Recall 0.978	Accuracy 0.987

Table A.9: Results for GO with SwissProt 1/4, Proteome Analyst

The following results were obtained by Proteome Analyst on selected Gene Ontology categories over a random selection of 25912 sequences from the SwissProt database. Coverage is 0.994.

Protein Class	Results TP / FP	FN / TN	NPP / NPN	Performance Precision	Recall	Accuracy
binding (0005488)	11962 / 601	903 / 12299	82 / 65	0.952	0.924	0.936
catalytic activity (0003824)	14927 / 307	393 / 10138	20 / 127	0.980	0.973	0.967
hydrolase activity (0016787)	4170 / 582	154 / 20859	23 / 124	0.878	0.959	0.966
lyase activity (0016829)	1398 / 183	41 / 24143	5 / 142	0.884	0.968	0.986
metal ion binding (0046872)	2343 / 780	174 / 22468	10 / 137	0.750	0.927	0.958
nucleic acid binding (0003676)	5137 / 693	178 / 19757	20 / 127	0.881	0.963	0.961
nucleotide binding (0000166)	3952 / 182	94 / 21537	8 / 139	0.956	0.975	0.984
oxidoreductase activity (0016491)	2674 / 394	44 / 22653	14 / 133	0.872	0.979	0.977
signal transducer activity (0004871)	1883 / 525	107 / 23250	38 / 109	0.782	0.929	0.970
structural molecule activity (0005198)	1902 / 60	65 / 23738	19 / 128	0.969	0.958	0.990
transferase activity (0016740)	4357 / 251	124 / 21033	7 / 140	0.946	0.971	0.980
transporter activity (0005215)	3470 / 570	124 / 21601	10 / 137	0.859	0.963	0.968
OVERALL	58175 / 5128	2401 / 243476	295 / 1469	0.919	0.956	0.970

Table A.10: Results for K^+ Channels, HMMer

Protein Class	Results				Performance		
Kv1	TP	23	FN	0	Precision	Recall	Accuracy
	FP	0	TN	49	1.000	1.000	1.000
Kv2	TP	14	FN	0	Precision	Recall	Accuracy
	FP	0	TN	58	1.000	1.000	1.000
Kv3	TP	17	FN	0	Precision	Recall	Accuracy
	FP	0	TN	41	1.000	1.000	1.000
Kv4	TP	18	FN	0	Precision	Recall	Accuracy
	FP	0	TN	54	1.000	1.000	1.000
OVERALL	TP	72	FN	0	Precision	Recall	Accuracy
	FP	0	TN	202	1.000	1.000	1.000

Table A.11: Results for Gram- Subcell, HMMer

Protein Class	Results				Performance		
cytoplasm	TP	2057	FN	430	Precision	Recall	Accuracy
	FP	430	TN	1043	0.827	0.827	0.783
extracellular	TP	56	FN	148	Precision	Recall	Accuracy
	FP	148	TN	3608	0.275	0.275	0.925
inner membrane	TP	110	FN	496	Precision	Recall	Accuracy
	FP	496	TN	2858	0.182	0.182	0.749
outer membrane	TP	127	FN	161	Precision	Recall	Accuracy
	FP	161	TN	3511	0.441	0.441	0.919
periplasm	TP	27	FN	402	Precision	Recall	Accuracy
	FP	402	TN	3129	0.063	0.063	0.797
OVERALL	TP	2377	FN	1637	Precision	Recall	Accuracy
	FP	1637	TN	14149	0.592	0.592	0.835

Table A.12: Results for K^+ Channels, Pfam and PROSITE with Naïve Bayes

Coverage is 100%. 20 PROSITE patterns. 379 Pfam families.

Protein Class	Results				Performance		
Kv1	TP	23	FN	0	Precision	Recall	Accuracy
	FP	1	TN	53	0.960	1.000	0.987
Kv2	TP	17	FN	0	Precision	Recall	Accuracy
	FP	0	TN	60	1.000	1.000	1.000
Kv3	TP	17	FN	0	Precision	Recall	Accuracy
	FP	0	TN	60	1.000	1.000	1.000
Kv4	TP	19	FN	0	Precision	Recall	Accuracy
	FP	0	TN	58	1.000	1.000	1.000
OVERALL	TP	76	FN	0	Precision	Recall	Accuracy
	FP	1	TN	231	0.987	1.000	0.997

Table A.13: Results for Gram- Subcell, Pfam and PROSITE with Naïve Bayes

Coverage is 97.2%. No predictions are done on proteins without pattern matches.

Protein Class	Results			Performance		
cytoplasm	TP 2284	FN 182	NPP 21	Precision	Recall	Accuracy
	FP 13	TN 1371	NPN 89	0.994	0.918	0.923
extracellular	TP 77	FN 110	NPP 17	Precision	Recall	Accuracy
	FP 40	TN 3623	NPN 93	0.658	0.377	0.934
inner membrane	TP 450	FN 133	NPP 23	Precision	Recall	Accuracy
	FP 150	TN 3117	NPN 87	0.750	0.743	0.901
outer membrane	TP 144	FN 127	NPP 17	Precision	Recall	Accuracy
	FP 71	TN 3508	NPN 93	0.670	0.500	0.922
periplasm	TP 230	FN 158	NPP 41	Precision	Recall	Accuracy
	FP 85	TN 3377	NPN 69	0.730	0.536	0.911
OVERALL	TP 3185	FN 710	NPP 119	Precision	Recall	Accuracy
	FP 359	TN 14996	NPN 550	0.899	0.793	0.913

Table A.14: Results for GO with SwissProt 1/4, Pfam and PROSITE with Naïve Bayes

Coverage is 96.1% for each class (although the positives and negatives not covered varies by class). Combined 5-fold cross-validation training and classification was about 240s per class. NOTE: these results need to be rerun with removal of the ones wich we can't say anything about. find the numbers of positives and negatives left out.

Protein Class	Results TP	FN	NPP	Performance		
	FP	TN	NPN			
binding (0005488)	11416	1068	463	Precision	Recall	Accuracy
	508	11910	547	0.957	0.882	0.900
catalytic activity (0003824)	14609	302	468	Precision	Recall	Accuracy
	555	9436	542	0.963	0.950	0.928
hydrolase activity (0016787)	3730	480	137	Precision	Recall	Accuracy
	364	20328	873	0.911	0.858	0.928
lyase activity (0016829)	952	467	25	Precision	Recall	Accuracy
	266	23217	985	0.782	0.659	0.933
metal ion binding (0046872)	1874	612	41	Precision	Recall	Accuracy
	544	21872	969	0.775	0.742	0.916
nucleic acid binding (0003676)	4637	513	185	Precision	Recall	Accuracy
	380	19372	825	0.924	0.869	0.927
nucleotide binding (0000166)	3631	298	125	Precision	Recall	Accuracy
	249	20724	885	0.936	0.896	0.940
oxidoreductase activity (0016491)	2338	301	93	Precision	Recall	Accuracy
	361	21902	917	0.866	0.856	0.935
signal transducer activity (0004871)	1544	387	97	Precision	Recall	Accuracy
	389	22582	913	0.799	0.761	0.931
structural molecule activity (0005198)	1760	175	51	Precision	Recall	Accuracy
	328	22639	959	0.843	0.886	0.942
transferase activity (0016740)	3890	440	158	Precision	Recall	Accuracy
	403	20169	852	0.906	0.867	0.928
transporter activity (0005215)	3056	399	149	Precision	Recall	Accuracy
	392	21055	861	0.886	0.848	0.930
OVERALL	53437	5442	1992	Precision	Recall	Accuracy
	4739	235206	10128	0.919	0.878	0.928

Table A.15: Results for K^+ Channels, Pfam and PROSITE with SVM

Coverage is 100%. 20 PROSITE patterns. 379 Pfam families.

Protein Class	Results				Performance		
Kv1	TP 23	FN 0			Precision	Recall	Accuracy
	FP 1	TN 53			0.958	1.000	0.987
Kv2	TP 15	FN 2			Precision	Recall	Accuracy
	FP 0	TN 60			1.000	0.882	0.974
Kv3	TP 17	FN 0			Precision	Recall	Accuracy
	FP 0	TN 60			0.944	1.000	0.987
Kv4	TP 19	FN 0			Precision	Recall	Accuracy
	FP 0	TN 58			1.000	1.000	1.000
OVERALL	TP 74	FN 2			Precision	Recall	Accuracy
	FP 1	TN 231			0.987	0.974	0.990

Table A.16: Results for Gram- Subcell, Pfam and PROSITE with SVM

Coverage is 97.2%. No predictions are done on proteins without pattern matches.

Protein Class	Results						Performance		
cytoplasm	TP 2297	FN 169	NPP 21				Precision	Recall	Accuracy
	FP 29	TN 1355	NPN 89				0.988	0.924	0.922
extracellular	TP 143	FN 44	NPP 17				Precision	Recall	Accuracy
	FP 16	TN 3647	NPN 93				0.899	0.701	0.957
inner membrane	TP 481	FN 102	NPP 23				Precision	Recall	Accuracy
	FP 14	TN 3253	NPN 87				0.972	0.794	0.943
outer membrane	TP 239	FN 32	NPP 17				Precision	Recall	Accuracy
	FP 9	TN 3570	NPN 93				0.964	0.830	0.962
periplasm	TP 311	FN 77	NPP 41				Precision	Recall	Accuracy
	FP 32	TN 3430	NPN 69				0.907	0.725	0.945
OVERALL	TP 3471	FN 424	NPP 119				Precision	Recall	Accuracy
	FP 100	TN 15255	NPN 431				0.972	0.865	0.946

Table A.17: Results for GO with SwissProt 1/4, Pfam and PROSITE with SVM

Coverage is 96.1% for each class (although the positives and negatives not covered varies by class). Combined 5-fold cross-validation training and classification was about 240s per class. NOTE: these results need to be rerun with removal of the ones wich we can't say anything about. find the numbers of positives and negatives left out.

Protein Class	Results TP	FN	NPP	Performance		
	FP	TN	NPN			
binding (0005488)	11603	881	463	Precision	Recall	Accuracy
	465	11953	547	0.961	0.896	0.909
catalytic activity (0003824)	14658	253	468	Precision	Recall	Accuracy
	587	9404	542	0.961	0.953	0.929
hydrolase activity (0016787)	3871	339	137	Precision	Recall	Accuracy
	165	20527	873	0.959	0.890	0.942
lyase activity (0016829)	1312	107	25	Precision	Recall	Accuracy
	48	23435	985	0.965	0.909	0.955
metal ion binding (0046872)	2046	440	41	Precision	Recall	Accuracy
	215	22201	969	0.905	0.810	0.936
nucleic acid binding (0003676)	4763	387	185	Precision	Recall	Accuracy
	185	19567	825	0.963	0.893	0.939
nucleotide binding (0000166)	3792	137	125	Precision	Recall	Accuracy
	74	20899	885	0.981	0.935	0.953
oxidoreductase activity (0016491)	2472	167	93	Precision	Recall	Accuracy
	108	22155	917	0.958	0.905	0.950
signal transducer activity (0004871)	1682	249	97	Precision	Recall	Accuracy
	51	22920	913	0.971	0.829	0.949
structural molecule activity (0005198)	1856	79	51	Precision	Recall	Accuracy
	19	22948	959	0.990	0.935	0.957
transferase activity (0016740)	4028	302	158	Precision	Recall	Accuracy
	122	20450	852	0.971	0.898	0.945
transporter activity (0005215)	3135	320	149	Precision	Recall	Accuracy
	87	21360	861	0.973	0.870	0.945
OVERALL	55218	3661	1992	Precision	Recall	Accuracy
	2126	237819	10128	0.963	0.907	0.942

Table A.18: Results for K^+ Channels, bPST

Coverage is 100%. Compare with BLAST nearest-neighbor results in Table A.4. These results were obtained using Bejerano's published parameters.

Protein Class	Results				Performance		
Kv1	TP	24	FN	0	Precision	Recall	Accuracy
	FP	0	TN	53	1.000	1.000	1.000
Kv2	TP	17	FN	0	Precision	Recall	Accuracy
	FP	0	TN	60	1.000	1.000	1.000
Kv3	TP	17	FN	0	Precision	Recall	Accuracy
	FP	0	TN	45	1.000	1.000	1.000
Kv4	TP	19	FN	0	Precision	Recall	Accuracy
	FP	0	TN	58	1.000	1.000	1.000
OVERALL	TP	77	FN	0	Precision	Recall	Accuracy
	FP	0	TN	216	1.000	1.000	1.000

Table A.19: Results for Gram- Subcell, bPST

Protein Class	Results				Performance		
cytoplasm	TP	2176	FN	311	Precision	Recall	Accuracy
	FP	311	TN	1162	0.875	0.875	0.843
extracellular	TP	128	FN	76	Precision	Recall	Accuracy
	FP	76	TN	3680	0.627	0.627	0.962
inner membrane	TP	446	FN	160	Precision	Recall	Accuracy
	FP	160	TN	3194	0.736	0.736	0.919
outer membrane	TP	203	FN	85	Precision	Recall	Accuracy
	FP	85	TN	3587	0.705	0.705	0.957
periplasm	TP	244	FN	185	Precision	Recall	Accuracy
	FP	185	TN	3346	0.569	0.569	0.907
OVERALL	TP	3197	FN	817	Precision	Recall	Accuracy
	FP	817	TN	14969	0.796	0.796	0.917

Table A.20: Results for GO with SwissProt 1/4, bPST

Protein Class	Results TP / FP	FN / TN	NPP / NPN	Performance		
binding (0005488)	8029	4918	-	Precision	Recall	Accuracy
	4918	8047	-	0.620	0.620	0.620
catalytic activity (0003824)	10389	4990	-	Precision	Recall	Accuracy
	4990	5543	-	0.676	0.676	0.615
hydrolase activity (0016787)	1461	2886	-	Precision	Recall	Accuracy
	2886	18679	-	0.336	0.336	0.777
lyase activity (0016829)	770	674	-	Precision	Recall	Accuracy
	674	23794	-	0.533	0.533	0.948
metal ion binding (0046872)	992	1535	-	Precision	Recall	Accuracy
	1535	21850	-	0.393	0.393	0.882
nucleic acid binding (0003676)	2797	2538	-	Precision	Recall	Accuracy
	2538	18039	-	0.524	0.524	0.804
nucleotide binding (0000166)	2020	2034	-	Precision	Recall	Accuracy
	2034	19824	-	0.498	0.498	0.843
oxidoreductase activity (0016491)	1334	1398	-	Precision	Recall	Accuracy
	1398	21782	-	0.488	0.488	0.892
signal transducer activity (0004871)	945	1083	-	Precision	Recall	Accuracy
	1083	22801	-	0.466	0.466	0.916
structural molecule activity (0005198)	1432	554	-	Precision	Recall	Accuracy
	554	23372	-	0.721	0.721	0.957
transferase activity (0016740)	1797	2691	-	Precision	Recall	Accuracy
	2691	18733	-	0.400	0.400	0.792
transporter activity (0005215)	1843	1761	-	Precision	Recall	Accuracy
	1761	20547	-	0.511	0.511	0.864
OVERALL	33809	27062	-	Precision	Recall	Accuracy
	27062	223011	-	0.555	0.555	0.826

Table A.21: Results for K^+ Channels, Global ePST at Isopoint

Coverage is 100%.

Protein Class	Results				Performance		
Kv1	TP	24	FN	0	Precision	Recall	Accuracy
	FP	0	TN	53	1.000	1.000	1.000
Kv2	TP	17	FN	0	Precision	Recall	Accuracy
	FP	0	TN	60	1.000	1.000	1.000
Kv3	TP	17	FN	0	Precision	Recall	Accuracy
	FP	0	TN	60	1.000	1.000	1.000
Kv4	TP	19	FN	0	Precision	Recall	Accuracy
	FP	0	TN	58	1.000	1.000	1.000
OVERALL	TP	77	FN	0	Precision	Recall	Accuracy
	FP	0	TN	231	1.000	1.000	1.000

Table A.22: Results for Gram- Subcell, Global ePST at Isopoint

Coverage is 100%.

Protein Class	Results				Performance		
cytoplasm	TP	2285	FN	202	Precision	Recall	Accuracy
	FP	202	TN	1271	0.919	0.919	0.898
extracellular	TP	138	FN	66	Precision	Recall	Accuracy
	FP	66	TN	3690	0.676	0.676	0.967
inner membrane	TP	455	FN	151	Precision	Recall	Accuracy
	FP	151	TN	3203	0.751	0.751	0.924
outer membrane	TP	227	FN	61	Precision	Recall	Accuracy
	FP	61	TN	3611	0.788	0.788	0.969
periplasm	TP	301	FN	128	Precision	Recall	Accuracy
	FP	128	TN	3403	0.702	0.702	0.935
OVERALL	TP	3406	FN	608	Precision	Recall	Accuracy
	FP	608	TN	15178	0.849	0.849	0.939

Table A.23: Results for GO with SwissProt 1/4, Global ePST at Isopoint

Coverage is 100%.

Protein Class	Results TP	FN	NPP	Performance		
	FP	TN	NPN			
binding (0005488)	10384	2563	·	Precision	Recall	Accuracy
	2563	10402	·	0.802	0.802	0.802
catalytic activity (0003824)	12525	2854	·	Precision	Recall	Accuracy
	2854	7679	·	0.814	0.814	0.780
hydrolase activity (0016787)	2917	1430	·	Precision	Recall	Accuracy
	1430	20135	·	0.671	0.671	0.890
lyase activity (0016829)	1107	337	·	Precision	Recall	Accuracy
	337	24131	·	0.767	0.767	0.974
metal ion binding (0046872)	1740	787	·	Precision	Recall	Accuracy
	787	22598	·	0.689	0.689	0.939
nucleic acid binding (0003676)	3913	1422	·	Precision	Recall	Accuracy
	1422	19155	·	0.733	0.733	0.890
nucleotide binding (0000166)	3141	913	·	Precision	Recall	Accuracy
	913	20945	·	0.775	0.775	0.930
oxidoreductase activity (0016491)	2109	623	·	Precision	Recall	Accuracy
	623	22557	·	0.772	0.772	0.952
signal transducer activity (0004871)	1402	626	·	Precision	Recall	Accuracy
	626	23258	·	0.691	0.691	0.952
structural molecule activity (0005198)	1635	351	·	Precision	Recall	Accuracy
	351	23575	·	0.823	0.823	0.973
transferase activity (0016740)	3210	1278	·	Precision	Recall	Accuracy
	1278	20146	·	0.715	0.715	0.901
transporter activity (0005215)	2476	1128	·	Precision	Recall	Accuracy
	1128	21180	·	0.687	0.687	0.913
OVERALL	46559	14312	·	Precision	Recall	Accuracy
	14312	235761	·	0.765	0.765	0.908

Table A.24: Results for K^+ Channels, ePST at Isopoint

Coverage is 100%.

Protein Class	Results				Performance		
Kv1	TP	23	FN	1	Precision	Recall	Accuracy
	FP	1	TN	52	0.958	0.958	0.974
Kv2	TP	16	FN	1	Precision	Recall	Accuracy
	FP	1	TN	59	0.941	0.941	0.974
Kv3	TP	16	FN	1	Precision	Recall	Accuracy
	FP	1	TN	59	0.941	0.941	0.974
Kv4	TP	19	FN	0	Precision	Recall	Accuracy
	FP	0	TN	58	1.000	1.000	1.000
OVERALL	TP	74	FN	3	Precision	Recall	Accuracy
	FP	3	TN	228	0.961	0.961	0.981

Table A.25: Results for Gram- Subcell, ePST at Isopoint

Coverage is 100%.

Protein Class	Results				Performance		
cytoplasm	TP	2240	FN	247	Precision	Recall	Accuracy
	FP	247	TN	1226	0.901	0.901	0.875
extracellular	TP	136	FN	68	Precision	Recall	Accuracy
	FP	68	TN	3688	0.667	0.667	0.966
inner membrane	TP	417	FN	189	Precision	Recall	Accuracy
	FP	189	TN	3165	0.688	0.688	0.905
outer membrane	TP	226	FN	62	Precision	Recall	Accuracy
	FP	62	TN	3610	0.785	0.785	0.969
periplasm	TP	276	FN	153	Precision	Recall	Accuracy
	FP	153	TN	3378	0.643	0.643	0.923
OVERALL	TP	3295	FN	719	Precision	Recall	Accuracy
	FP	719	TN	15067	0.821	0.821	0.927

Table A.26: Results for GO with SwissProt 1/4, ePST at Isopoint

Coverage is 100%.

Protein Class	Results TP / FP	FN / TN	NPP / NPN	Performance		
binding (0005488)	10404	2543	·	Precision	Recall	Accuracy
	2543	10422	·	0.804	0.804	0.804
catalytic activity (0003824)	12856	2523	·	Precision	Recall	Accuracy
	2523	8010	·	0.836	0.836	0.805
hydrolase activity (0016787)	3118	1229	·	Precision	Recall	Accuracy
	1229	20336	·	0.717	0.717	0.905
lyase activity (0016829)	1145	299	·	Precision	Recall	Accuracy
	299	24169	·	0.793	0.793	0.977
metal ion binding (0046872)	1803	724	·	Precision	Recall	Accuracy
	724	22661	·	0.713	0.713	0.944
nucleic acid binding (0003676)	3929	1406	·	Precision	Recall	Accuracy
	1406	19171	·	0.736	0.736	0.891
nucleotide binding (0000166)	3413	641	·	Precision	Recall	Accuracy
	641	21217	·	0.842	0.842	0.951
oxidoreductase activity (0016491)	2153	579	·	Precision	Recall	Accuracy
	579	22601	·	0.788	0.788	0.955
signal transducer activity (0004871)	1381	647	·	Precision	Recall	Accuracy
	647	23237	·	0.681	0.681	0.950
structural molecule activity (0005198)	1470	516	·	Precision	Recall	Accuracy
	516	23410	·	0.740	0.740	0.960
transferase activity (0016740)	3374	1114	·	Precision	Recall	Accuracy
	1114	20310	·	0.752	0.752	0.914
transporter activity (0005215)	2427	1177	·	Precision	Recall	Accuracy
	1177	21131	·	0.673	0.673	0.909
OVERALL	47473	13398	·	Precision	Recall	Accuracy
	13398	236675	·	0.780	0.780	0.914

Table A.27: Results for K^+ Channels, ePST at FMAX 40

Coverage is 100%. Compare with BLAST nearest-neighbor results in Table A.4.

Protein Class	Results				Performance		
Kv1	TP 24	FN 0	FP 1	TN 52	Precision 0.960	Recall 1.000	Accuracy 0.987
Kv2	TP 17	FN 0	FP 1	TN 59	Precision 0.944	Recall 1.000	Accuracy 0.987
Kv3	TP 17	FN 0	FP 1	TN 59	Precision 0.944	Recall 1.000	Accuracy 0.987
Kv4	TP 19	FN 0	FP 0	TN 58	Precision 1.000	Recall 1.000	Accuracy 1.000
OVERALL	TP 77	FN 0	FP 3	TN 228	Precision 0.963	Recall 1.000	Accuracy 0.990

Table A.28: Results for Gram- Subcell, Pfam and PROSITE with ePST FMAX 40

Coverage is 100%.

Protein Class	Results				Performance		
cytoplasm	TP 2174	FN 313	FP 33	TN 1440	Precision 0.985	Recall 0.874	Accuracy 0.913
extracellular	TP 132	FN 72	FP 10	TN 3746	Precision 0.930	Recall 0.647	Accuracy 0.979
inner membrane	TP 417	FN 189	FP 144	TN 3210	Precision 0.743	Recall 0.688	Accuracy 0.916
outer membrane	TP 223	FN 65	FP 15	TN 3657	Precision 0.937	Recall 0.774	Accuracy 0.980
periplasm	TP 255	FN 174	FP 15	TN 3516	Precision 0.944	Recall 0.594	Accuracy 0.952
OVERALL	TP 3201	FN 813	FP 217	TN 15569	Precision 0.937	Recall 0.797	Accuracy 0.948

Table A.29: Results for GO with SwissProt 1/4, ePST at FMAX 40

Coverage is 100%. Compare with BLAST nearest-neighbor results in Table A.4.

Protein Class	Results (TP / FN / NPP; FP / TN / NPN)			Performance		
	TP	FN		Precision	Recall	Accuracy
binding (0005488)	9780	3167		0.880	0.755	0.826
	FP	TN				
	1330	11635				
catalytic activity (0003824)	TP	FN		Precision	Recall	Accuracy
	12581	2798		0.862	0.818	0.814
	FP	TN				
	2015	8518				
hydrolase activity (0016787)	TP	FN		Precision	Recall	Accuracy
	3008	1339		0.786	0.692	0.917
	FP	TN				
	821	20744				
lyase activity (0016829)	TP	FN		Precision	Recall	Accuracy
	1097	347		0.867	0.760	0.980
	FP	TN				
	169	24299				
metal ion binding (0046872)	TP	FN		Precision	Recall	Accuracy
	1689	838		0.805	0.668	0.952
	FP	TN				
	408	22977				
nucleic acid binding (0003676)	TP	FN		Precision	Recall	Accuracy
	3663	1672		0.865	0.687	0.913
	FP	TN				
	570	20007				
nucleotide binding (0000166)	TP	FN		Precision	Recall	Accuracy
	3340	714		0.882	0.824	0.955
	FP	TN				
	445	21413				
oxidoreductase activity (0016491)	TP	FN		Precision	Recall	Accuracy
	2040	692		0.921	0.747	0.967
	FP	TN				
	174	23006				
signal transducer activity (0004871)	TP	FN		Precision	Recall	Accuracy
	1301	727		0.771	0.642	0.957
	FP	TN				
	386	23498				
structural molecule activity (0005198)	TP	FN		Precision	Recall	Accuracy
	1407	579		0.838	0.708	0.967
	FP	TN				
	272	23654				
transferase activity (0016740)	TP	FN		Precision	Recall	Accuracy
	3168	1320		0.850	0.706	0.928
	FP	TN				
	558	20866				
transporter activity (0005215)	TP	FN		Precision	Recall	Accuracy
	2232	1372		0.845	0.619	0.931
	FP	TN				
	408	21900				
OVERALL	TP	FN		Precision	Recall	Accuracy
	45306	15565		0.857	0.744	0.926
	FP	TN				
	7556	242517				

Table A.30: Results for GO with SwissProt 1/4, ePST without Smoothing

Coverage is 100%.

Protein Class	Results			Performance		
	TP	FN	NPP			
	FP	TN	NPN			
binding (0005488)	TP 10083	FN 2864		Precision	Recall	Accuracy
	FP 1666	TN 11299		0.858	0.779	0.825
catalytic activity (0003824)	TP 12562	FN 2817		Precision	Recall	Accuracy
	FP 1903	TN 8630		0.868	0.817	0.818
hydrolase activity (0016787)	TP 3031	FN 1316		Precision	Recall	Accuracy
	FP 850	TN 20715		0.781	0.697	0.916
lyase activity (0016829)	TP 1124	FN 320		Precision	Recall	Accuracy
	FP 215	TN 24253		0.839	0.778	0.979
metal ion binding (0046872)	TP 1724	FN 803		Precision	Recall	Accuracy
	FP 479	TN 22906		0.783	0.682	0.951
nucleic acid binding (0003676)	TP 3794	FN 1541		Precision	Recall	Accuracy
	FP 720	TN 19857		0.840	0.711	0.913
nucleotide binding (0000166)	TP 3356	FN 698		Precision	Recall	Accuracy
	FP 466	TN 21392		0.878	0.828	0.955
oxidoreductase activity (0016491)	TP 2067	FN 665		Precision	Recall	Accuracy
	FP 226	TN 22954		0.901	0.757	0.966
signal transducer activity (0004871)	TP 1279	FN 749		Precision	Recall	Accuracy
	FP 352	TN 23532		0.784	0.631	0.958
structural molecule activity (0005198)	TP 1428	FN 558		Precision	Recall	Accuracy
	FP 344	TN 23582		0.806	0.719	0.965
transferase activity (0016740)	TP 3188	FN 1300		Precision	Recall	Accuracy
	FP 560	TN 20864		0.851	0.710	0.928
transporter activity (0005215)	TP 2226	FN 1378		Precision	Recall	Accuracy
	FP 446	TN 21862		0.833	0.618	0.930
OVERALL	TP 45862	FN 15009		Precision	Recall	Accuracy
	FP 8227	TN 241846		0.848	0.753	0.925

Table A.31: Comparison over *pkinase* Pfam Family

Protein Class	Results				Performance			
HMMer	TP	374	FN	3	Precision	Recall	Accuracy	Coverage
	FP	3	TN	25532	0.992	0.992	1.000	1.000
bPST	TP	195	FN	182	Precision	Recall	Accuracy	Coverage
	FP	182	TN	25353	0.517	0.517	0.986	1.000
ePST Global	TP	224	FN	153	Precision	Recall	Accuracy	Coverage
	FP	153	TN	25382	0.594	0.594	0.988	1.000
ePST Local 40	TP	254	FN	123	Precision	Recall	Accuracy	Coverage
	FP	123	TN	25412	0.674	0.674	0.991	1.000
ePST Local 200	TP	281	FN	96	Precision	Recall	Accuracy	Coverage
	FP	96	TN	25439	0.745	0.745	0.993	1.000

Table A.32: Comparison over *metallothio* Pfam Family

Protein Class	Results				Performance			
HMMer	TP	39	FN	0	Precision	Recall	Accuracy	Coverage
	FP	0	TN	25873	1.000	1.00 0	1.000	1.000
bPST	TP	38	FN	1	Precision	Recall	Accuracy	Coverage
	FP	1	TN	25872	0.974	0.974	1.000	1.000
ePST Global	TP	35	FN	4	Precision	Recall	Accuracy	Coverage
	FP	4	TN	25869	0.897	0.897	1.000	1.000
ePST Local 40	TP	35	FN	4	Precision	Recall	Accuracy	Coverage
	FP	4	TN	25869	0.897	0.897	1.000	1.000

Table A.33: Comparison of Mixed *pkinase* and *metallothio* Pfam Families

Protein Class	Results				Performance			
HMM	TP	377	FN	39	Precision	Recall	Accuracy	Coverage
	FP	99	TN	25457	0.906	0.906	0.997	1.000
bPST	TP	230	FN	186	Precision	Recall	Accuracy	Coverage
	FP	186	TN	25310	0.553	0.553	0.986	1.000
ePST Global	TP	270	FN	146	Precision	Recall	Accuracy	Coverage
	FP	146	TN	25350	0.649	0.649	0.989	1.000
ePST Local 40	TP	302	FN	114	Precision	Recall	Accuracy	Coverage
	FP	114	TN	25382	0.726	0.726	0.991	1.000

Appendix B

Supplementary Tables and Figures

Figure B.1: Example Training Sequences - PSSM

```
>sequence1
QIKDLLVSSSTDLDTTLVLRENVATLPAEKMKPAINDAFE
KWPATESFHVTILELKYFQESKPVMPQMMCNS
>sequence2
RRVKVYLPQMKIEEKYNLTSVLMALGMTDLFIPSANLTFI
ADMFTEETTERNSKISQAGSSAESLIGVIEDIKHSP
>sequence3
ISEEYISYGGEKKILAIQGALEKALRWASGESFIELSNHK
FDRMFINDFMTHEKLPAERSSAKFRRFT
>sequence4
AKLAEQAERYDDNLLSVAYKNVVGARRSSWRVISSIEQKT
ERNEKKQQMGKEYREKIEAELQDICNDVLELLDKFINDMF
THEKLPATRNSYLIPNRSQPESKVFYLKMKGDYFRYLSEV
ASGDNKQTVSNSQQAYQEAFEISKKEMQPT
>sequence5
MITILEKISAIESEMARTQKNKATSAHLGGGGTGEAGFEV
AKTGDARVGFVGFPSVGKSTLLSNLAGVMFIADANTHELR
PATNSYSEVAAYEFTTLTTVPGCIKYKGAKIQLLDLPGII
EGAKDGKGRGRQVIAVARTC
>sequence6
QFINKNKTAETSPAHLERNGGGGTGEAGGVGFPSVGKSTL
SNLGVFNNDANHELRAERNSYSEAAYEFTTLTTVPGCIKY
KGAKIQLLDLPGIIEGAKDGKGRGRQVIAVART
>sequence3
ISEEYISYGGEKDINEKILAIQGALEKALRWASGESFIEL
SNHKFDRMDINDRKTHEKLSSAKFRRFT
```

Figure B.2: Example Training Sequences - Markov Chain

```
>positive 1
KPFFINDAFTHEKWPATTERNSF
>positive 2
LTFINDMFTHEEDPATTERNSK
>positive 3
RMFINDRKTHEKLPATTERNSSA
>positive 4
ELLDKFINDMKTHEKLPATTERNSYLIP
>positive 5
LAGVFINDANTHELRPATTERNSYS
>negative 1
HVTILELKYFQESKPVMPQMMCNSQIKDLLVSSSTDLDTT
LVLRENVATLPAEKM
>negative 2
RRVKVYLPQMKIEEKYNLTSVLMALGMTDLFIPSANISQA
GSSAESLIGVIEDIKHSP
>negative 3
ISEEYISYGGEKKILAIQGALEKALRWASGESFIELSNHK
FDKFRRFT
>negative 4
AKLAEQAERYDDNLLSVAYKNVVGARRSSWRVISSIEQKT
ERNEKKQQMGKEYREKIEAELQDICNDVLNRSQPESKVFY
LKMKGDYFRYLSEVASGDNKQTVSNSQQAYQEAFEISKKE
MQPT
>negative 5
MITILEKISAIESEMARTQKNKATSAHLGGGGTGEAGFEV
AKTGDARVGFVGFPSVGKSTLLSNEVAAYEFTTLTTVPGC
IKYKGAKIQLLDLPGIIEGAKDGKGRGRQVIAVARTC
```

Table B.1: The 20 Amino Acids and Mappings to Smaller Alphabets.

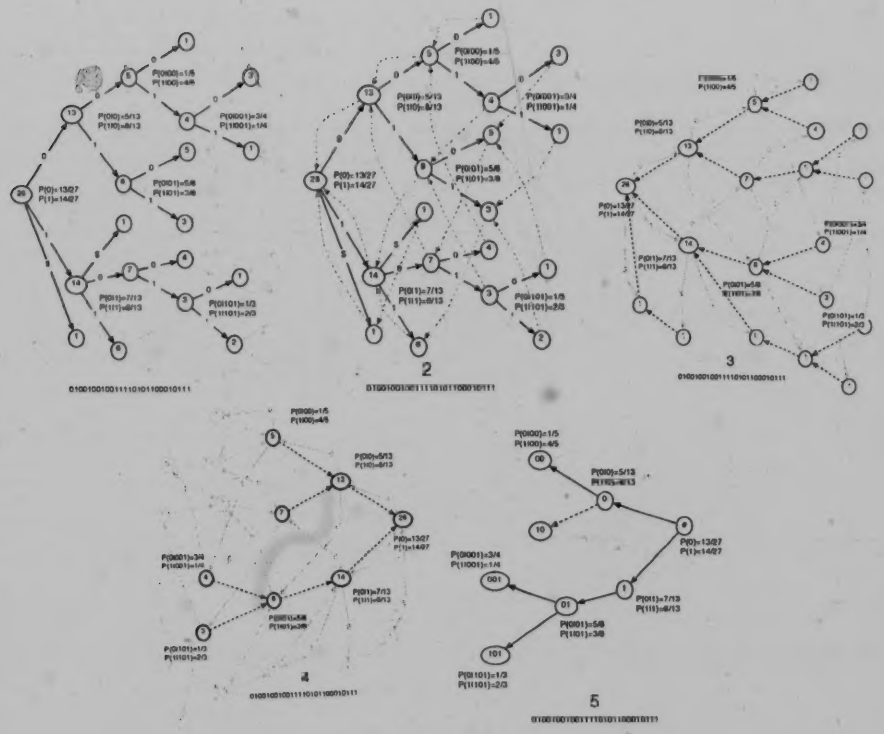
From Murphy *et al* [72, 45]. For the 10 letter alphabet the L, S, F, E, and K symbols indicate the large hydrophibic, polar, hydrophobic/aromatic sidechained, charged/polar, and long-chain positively charged amino acids respectively. For the 2 letter alphabet, P and E indicate the hydrophobic and hydrophilic amino acids respectively.

Amino Acid	Code	10 Letter Alphabet	2 Letter Alphabet
Alanine	A	A	P
Arginine	R	K	E
Asparagine	N	E	E
Aspartic acid (Aspartate)	D	E	E
Cystine	C	C	P
Glutamine	Q	E	E
Glutamic acid (Glutamate)	E	E	E
Glycine	G	G	P
Histidine	H	H	E
Isoleucine	I	L	P
Leucine	L	L	P
Lysine	K	K	E
Methionine	M	L	P
Phenylalanine	F	F	P
Proline	P	P	P
Serine	S	S	P
Threonine	T	S	P
Tryptophan	W	F	P
Tyrosine	Y	F	P
Valine	V	L	P

Figure B.3: Mapped Example Training Sequences

```
>positive 1
EAFFLEEAFAEEEFAAAAEEEAF
>positive 2
LAFLEELFAEEEEAAAAEEEAE
>positive 3
ELFLEEEEAEEELAAAAEEEAAA
>positive 4
ELLEEFLEELEAEEELAAAAEEEAFLLA
>positive 5
LAALFLEEAEAEELEAAAAEEEAFA
>negative 1
ELALLELEFFEEAEALLAELLLEAELEELLLAAAAELEAA
LLLEEELAALAAEEL
>negative 2
EELELFLAELELEEEFELAALLLALALAELFLAAAELAEA
AAAAEALLALLEELEEAA
>negative 3
LAEEFLAFAAEEELLALEAALEEALEFAAAEAFLELAEEE
FEEFEEFA
>negative 4
AELAEEAEEFEEELLALAFEELLAAEEAAFELLAALEEEA
EEEEEEEELAEEFEEELEAELEELLEELLEEAEAEAELFF
LELEAEFFEFLAELAAAEEEEALAEAEEAFEEAFELAEEE
LEAA
>negative 5
LLALLEELAALEAELAEAEEEEAAAAELAAAAAAEAAFEL
AEAAEAELAFLAFAALAEAALLAEELAAFEFAALAALAAL
LEFEAAELELLELAALLEAAEEAEAEAEELLALAEAL
```

Figure B.4: Conversion of an ePST to a bPST



Appendix C

Backoff Smoothing

Assume the following definition of the smoothed conditional probabilities, we wish to obtain a definition of the parameter α.

$$P_s(x_i|x_{i-1},\ldots,x_{i-k}) = \begin{cases} \dfrac{C(x_{i-k}\ldots x_{i-1}x_i)+d}{C(x_{i-k}\ldots x_{i-1})+d|\Sigma|} & \text{if } C(x_{i-k}\ldots x_i)>0 \\ \alpha(x_{i-k}\ldots x_{i-1}) & \\ \cdot P_s(x_i|x_{i-1},\ldots,x_{i-k+1}) & \text{otherwise} \end{cases} \quad \text{(C.1)}$$

Since the conditional probabilities for all values of x_i must sum to 1, we can calculate Δ, the total amount discounted by the pseudocount d from the non-zero cases. We need a definition for α.

$$\begin{aligned} 1 &= \sum_{a\in\Sigma} P_s(a|x_{i-k}\ldots x_{i-1}) && \text{(C.2)} \\ &= \sum_{a\in\Sigma, C(x_{i-k}\ldots x_{i-1}a)>0} P_s(a|x_{i-k}\ldots x_{i-1}) \\ &\quad + \sum_{a\in\Sigma, C(x_{i-k}\ldots x_{i-1}a)=0} P_s(a|x_{i-k}\ldots x_{i-1}) && \text{(C.3)} \\ &= \sum_{a\in\Sigma, C(x_{i-k}\ldots x_{i-1}a)>0} P_s(a|x_{i-k}\ldots x_{i-1}) + \Delta && \text{(C.4)} \end{aligned}$$

We now see delta as the amount which has been discounted from the non-zero cases and donated to the zero cases.

$$\begin{aligned} \Delta &= \sum_{a\in\Sigma, C(x_{i-k}\ldots x_{i-1}a)=0} P_s(a|x_{i-k}\ldots x_{i-1}) && \text{(C.5)} \\ &= \sum_{a\in\Sigma, C(x_{i-k}\ldots x_{i-1}a)=0} \alpha(x_{i-k}\ldots x_{i-1}) P_s(a|x_{i-k+1}\ldots x_{i-1}) && \text{(C.6)} \end{aligned}$$

$$= \alpha(x_{i-k} \ldots x_{i-1}) \sum_{a \in \Sigma, C(x_{i-k} \ldots x_{i-1} a)=0} P_s(a|x_{i-k+1} \ldots x_{i-1}) \quad \text{(C.7)}$$

We can rearrange this to get a value for α.

$$\alpha(x_{i-k} \ldots x_{i-1}) = \frac{\Delta}{\sum_{a \in \Sigma, C(x_{i-k} \ldots x_{i-1} a)=0} P_s(a|x_{i-k+1} \ldots x_{i-1})} \quad \text{(C.8)}$$

This is expressed in terms of the zero counts. We can express the denominator in terms of the non-zero counts.

$$\alpha(x_{i-k} \ldots x_{i-1}) = \frac{\Delta}{1 - \sum_{a \in \Sigma, C(x_{i-k} \ldots x_{i-1} a)>0} P_s(a|x_{i-k+1} \ldots x_{i-1})} \quad \text{(C.9)}$$

We also know from above the Δ can be expressed in terms of non-zero counts.

$$\Delta = 1 - \sum_{a \in \Sigma, C(x_{i-k} \ldots x_{i-1} a)>0} P_s(a|x_{i-k} \ldots x_{i-1}) \quad \text{(C.10)}$$

We can replace this in the above equation to get a full definition of α.

$$\alpha(x_{i-k} \ldots x_{i-1}) = \frac{1 - \sum_{a \in \Sigma, C(x_{i-k} \ldots x_{i-1} a)>0} P_s(a|x_{i-k} \ldots x_{i-1})}{1 - \sum_{a \in \Sigma, C(x_{i-k} \ldots x_{i-1} a)>0} P_s(a|x_{i-k+1} \ldots x_{i-1})} \quad \text{(C.11)}$$

$$= \frac{1 - \sum_{a \in \Sigma, C(x_{i-k} \ldots x_{i-1} a)>0} \frac{C(x_{i-k} \ldots x_{i-1} x_i) + d}{C(x_{i-k} \ldots x_{i-1}) + d|\Sigma|}}{1 - \sum_{a \in \Sigma, C(x_{i-k} \ldots x_{i-1} a)>0} \frac{C(x_{i-k+1} \ldots x_{i-1} x_i) + d}{C(x_{i-k+1} \ldots x_{i-1}) + d|\Sigma|}} \quad \text{(C.12)}$$